Software Engineering Institute

# FloCon 2006 Proceedings

January 2006

CERT Program

http://www.sei.cmu.edu

CarnegieMellon

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| 1. REPORT DATE<br>**JAN 2006** | 2. REPORT TYPE | 3. DATES COVERED<br>**00-00-2006 to 00-00-2006** |

4. TITLE AND SUBTITLE
**FloCon 2006 Proceedings**

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
**Carnegie Mellon University ,Software Engineering Institute (SEI),Pittsburgh,PA,15213**

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **Same as Report (SAR)** | **205** | |

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

# IPFIX/PSAMP: What Future Standards can Offer to Network Security

Tanja Zseby[1], Elisa Boschi[2], Thomas Hirsch[1], Lutz Mark[1]
[1]*Fraunhofer Fokus*
*{zseby, ,hirsch, mark}@fokus.fraunhofer.de*
[2]*Hitachi Europe*
*elisa.boschi@hitachi-eu.com*

## Abstract

*Network security often requires the surveillance of the actual traffic in the network. Methods like signature-based attack detection or the detection of traffic anomalies require input from network measurements. The IETF currently standardizes the IP Flow Information Export (IPFIX) protocol for exporting flow information from routers and probes. The packet sampling (PSAMP) group extends the information model of IPFIX with the ability to report per packet information including parts of the payload. With this IPFIX and PSAMP provide valuable tools for detecting anomalies and security incidents in IP networks. Whereas the basic IPFIX and PSAMP documents are currently finalized, new drafts emerge that provide recommendations and IPFIX extensions. This paper shows how IPFIX and PSAMP can be used to support network security. Furthermore it is shown which extensions are useful and can provide further features for network security.*

## 1. IPFIX and PSAMP

IPFIX defines a format and a protocol for the export of flow information from routers or measurement probes [1]. IPIFX uses a push-based data export, from IPFIX exporters to IPFIX collectors, and can run over TCP, UDP and SCTP. Figure 1 shows the process of measurement and export of IPFIX and PSAMP data. Core functions are always part of the measurement process. Optional functions can be placed in the processing sequence for different operations like post processing or data selection.

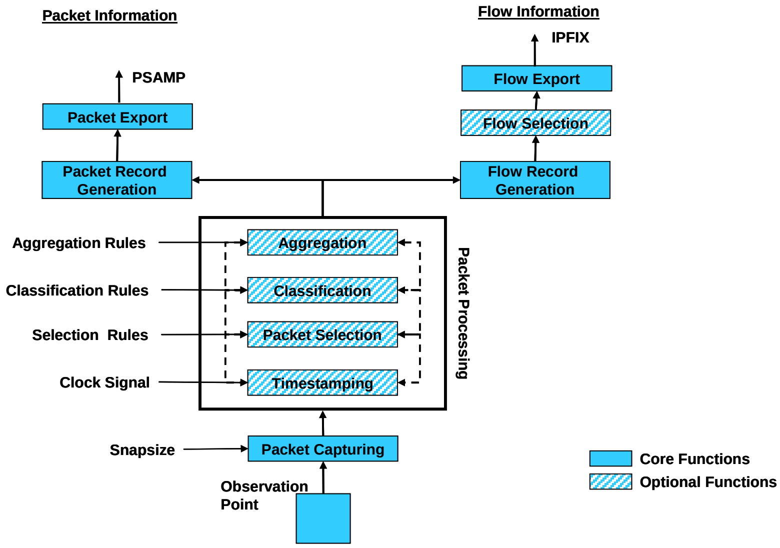


**Figure 0: Measurement Model**

[2] specifies observation point, flows, exporting and collecting process. The document also defines a metering process that consists of packet header capturing, timestamping, classifying, sampling and maintaining flow records. IPFIX Information Elements (IEs) for distinguishing flows and for reporting flow characteristics are defined in [3]. PSAMP extends the IPFIX information model by IEs for packet header and payload [4]. Furthermore it defines packet selection methods like filtering and sampling.

## 2. Metrics of Interest for Attack Detection

Network anomaly detection aims at discovering malicious behavior in a network by an analysis of the traffic profile. Using statistical means to detect unusual behavior patterns in the network or on the target machines, indicators for abuse or attacks are collected. While it can not be known in advance how precisely an attack will look like, experience has shown that certain information in packet headers provides better evidence than other. Also, some popular attack types can be easily detected by metering specific aspects of flow behavior.

Many detection algorithms base on very simple metrics. A successful detection of brute force attacks can be successfully done on data available from IPFIX. Early approaches like [5, 6] use simple flow definitions; and [7] successfully detects intrusions using only packet, ICMP packet and byte count using artificial immune systems. Most systems however face two difficulties: on the one hand, more advanced network intrusions are not easily detected by observing lower layers only. On the other, regular network events may produce a legitimate anomaly. The most common example is the so-called "flash crowds", a sudden increases in traffic caused by a reference from another high-volume Internet service or news site. These events have to be distinguished from malicious attacks. Here, many algorithms rely on specialized metrics for popular types of attacks. The difference between TCP SYN and FIN packets for example is a clear indicator of ongoing SYN flooding attacks [8, 9].

Other general capabilities of interest for attack detection with IPFIX are: Flow separation by transport (e.g., TCP, UDP) or application layer (e.g., HTTP, FTP) protocols [10] or the retrieval of information from higher-level protocol headers such as TCP/IP [11] or information from MIB-II [12]. Further approaches use specialized statistics for attack detection that model real user behaviour more closely; [2] for example defines a question as any number of consecutive packets going from the client to the server. The number of questions (and answers resp.) per second is used as parameter for configuring self-organized maps. Finally, samples of the full payload information [13] allow further insight into transactions.

## 3. Measurement Requirements and what IPFIX and PSAMP can Offer

The detection of traffic anomalies requires passive measurements of the traffic in the network. IPFIX and PSAMP can be implemented on routers or probes and provide a standardized method to export flow and packet information from different points in the network. A variety of metrics are of interest for anomaly detection (see section 2). Currently IPFIX defines IEs for all IPv4 header fields (except checksum), the main IPv6 header fields (addresses, next header, flow label, etc.), the main transport header fields (UDP, TCP ports, sequence numbers, ICMP types), and some sub IP header fields (MAC addresses, MPLS labels, etc.). For reporting of flow statistics it defines a variety of counters (e.g. bytes, packets, delta and total counters), timestamps (flow start, end, duration) and basic statistics (min/max pktlength, min/max TTL, TCP flags, options). PSAMP extends the information model by adding IEs for reporting the full header and payload information. A useful information element for attack detection would be a counter to report the number of packets with specific flags (e.g. SYN, FIN) in a flow (e.g. to a specific destination address). This is currently not provided by IPFIX; IPFIX only supports the IE tcpCcontrolBits, which is a bitfield with all TCP flags, where bits are set if a particular flag was observed for the flow. Nevertheless, the information model can be easily extended to support this counter.

In order to detect unusual behavior at different granularities or timescales, traffic needs to be observed at different aggregation levels. IPFIX provides an extremely flexible flow definition; a flow is defined as a set of packets with common properties. Each property is defined as a result of applying a function to one or more packet header fields (e.g. destination IP address), to further packet properties (e.g. number of MPLS labels) or to values derived from packet treatment (e.g. output IF). IEs defined in [3] can be used as flow keys to distinguish flows.

The analysis of the connection status (e.g. for TCP connections) requires a mapping of both directions of a communication. IPFIX currently reports each direction of a flow separately. With some additional effort a mapping of both directions is possible without IPFIX extensions. A more efficient way that introduces IEs for forward and backward direction is discussed in [14].

Distributed metrics often outperform metrics collected at a single observation point (see e.g., [15], [16], [17]), therefore data from multiple observation points should be correlated. Using identical flow keys at the observation points provides a network-wide picture about the flow situation. Synchronized clocks at the involved observation points allow the calculation of time-related metrics like one-way delay. If packet data needs to be correlated packet arrival events need to be recognised at different observation points. This can be done based on the packet content (header and optionally payload). For this only fields should be used that are immutable during transport but highly variable between different packets. In case the packet content itself is not needed (e.g. when calculating delay etc.) a packet ID based on those fields can be generated and post processing will be based only on this ID (see [18], [19], [20]). This significantly reduces the traffic that needs to be reported.

Post-incident analysis (network forensics) requires the storage of past data. This is also useful for sharing information among providers and to provide training data with “normal” behavior. In [21] requirements for an IPFIX file format are discussed and existing solutions for storage of flow information are investigated. It is planned to propose an IPFIX file format based on this study.

The ability to calculate specific metrics (e.g. packet ratios, statistics, etc.) directly on the router is a desired feature, since even if it consumes processing power on routers, it increases the speed at which incidents can be detected. Furthermore the reporting of derived metrics requires fewer transport resources than the export of all raw data. A disadvantage is the inability to derive arbitrary other metrics. If one does not know what to look for one can apply different methods on captured raw data. This is not possible if only derived metrics are reported. The reporting of derived metrics can be realized by extending the information model with new IEs as described in [22]. IPFIX is a push-based protocol. Currently the sending of flow records is triggered by flow termination criteria (e.g. flow idle time, TCP FIN, etc.) or resource limitations (cache full). If attack detection metrics are calculated directly on the router thresholds on these metrics could be used to trigger flow export. This would allow to reduce flow export to only those cases were suspicious behavior was observed.

Re-configuration of measurement processes is useful to zoom in or out based on the actual situation. Since the IPFIX group wanted first to concentrate on the protocol, the configuration of IPFIX functions was out of scope. Now that the IPFIX protocol is finished, several proposals for IPFIX configuration emerged. A first draft for an IPIFX MIB was described in [23]. An XML data model for configuration of IPFIX processes was proposed in [24]. Furthermore the Next Steps in Signaling (NSIS) group proposed a draft for path-coupled dynamic configuration of metering entities [25]. This framework can be used to configure parameters for IPFIX processes. A further desired feature is cost efficiency. Resources can be reduced by using filtering or sampling techniques as described in [26]. [27] and [28] describe methods for aggregation and sharing of flow key information among data records.

An interoperation of measurement functions with AAA functions provides further features for network security [22]. AAA Functions may be able to map the traffic to specific users (e.g. by using the src address) and can stop network access for suspicious systems or users. Furthermore AAA provides secure channels to neighbor AAA servers and can inform neighbors about incidents or suspicious observations. Although most providers are still reluctant to information sharing, the ability to share information with neighbor domains is a useful feature. IPFIX provides the means to do that: TCP or SCTP can be used as transport protocol to ensure congestion-awareness and IPsec and TLS can be used as described in [1] to provide security features.

Table 1 summarizes the measurement requirements and shows how IPFIX, PSAMP and/or IPFIX extensions support specific features.

| **Measurement Requirement** | **IPFIX support** | **PSAMP support** | **IPFIX extensions** |
|---|---|---|---|
| Network-wide passive measurements | Passive flow measurements integrated in routers | Packet capturing integrated in routers | - |
| Different aggregation levels | Flexible flow definition | Packet selection methods | [27], [28] |
| Variety of metrics | IEs for flow statistics, extensible info model | IEs for packet capturing, extensible info model | New IEs can be easily added |
| Analysis of connections | TCP flags bitmap | Header and payload information | [14] |
| Correlation from multiple observation points | Header fields for packet ID generation | Header and payload info for packet ID generation | [22] |
| Storage of past data | - | - | [21] |
| Export of derived metrics | - | - | [22], further planned |
| (Re-)configurability | - | Configuration of packet selection methods | [23], [24], [25] |
| Cost efficiency | Aggregation, packet selection | Packet selection methods | [27], [28] |
| Link to AAA functions | - | - | [22] |
| Inter-domain data exchange | Standard format, congestion-aware (TCP, SCTP), secure (IPsec, TLS) | Standard format, congestion-aware(TCP, SCTP), secure (IPsec, TLS) | [22] |

**Table 1: IPFIX and PSAMP Support for Anomaly Detection**

## 5. Conclusions

IPFIX and PSAMP provide standardized measurement methods to support network security applications like attack and anomaly detection. A variety of relevant metrics can be derived from IPFIX and PSAMP data. Useful IPIFX extensions for correlation, aggregation and storage of IPIFX data have been proposed already within the IETF. Approaches for IPFIX configuration are underway.

Fraunhofer FOKUS has developed an open source IPIFX implementation. Besides the standard IPFIX IEs it supports proprietary IEs for reporting QoS metrics (loss, delay, jitter), TCP flag counters and packet IDs. The FOKUS IPFIX implementation is available at [29].

# References

[1] B. Claise (Editor), "IPFIX Protocol Specification", Internet Draft, work in progress, June 2006

[2] J. Quittek, T. Zseby, B. Claise, S. Zander, "Requirements for IP Flow Information Export", RFC3917, October 2004

[3] J. Quittek, S. Bryant, J. Meyer, "Information Model for IP Flow Information Export", Internet Draft, work in progress, July 2006

[4] T. Dietz, F. Dressler, G. Carle, B. Claise, "Information Model for Packet Sampling Exports", Internet Draft, work in progress, June 2006

[5] Information-theoretic measures for anomaly detection, Wenke Lee; Dong Xiang, IEEE Symposium on Security and Privacy, S&P 200. 14-16, May 2001, Pages:130 - 143

[6] P. Barford, J. Kline, D. Plonka, A. Ron, "A signal analysis of network traffic anomalies", Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement, 2002 Marseille, France. Pages: 71 - 8

[7] M. Ostaszewski, F. Seredynski, and P. Bouvry. "A nonself space approach to network anomaly detection". In 20th International Parallel and Distributed Processing Symposium (IPDPS), NIDISC, Rhodes, Greece, April 2006.

[8] R.B.Blazek, H.Kim, B.Rozovskii, A.Tartakovsky, "A novel approach to detection of DoS attacks via adaptive sequential and batch-sequential change-point methods", Proceedings of the 2001 IEEE Workshop on Information Assurance and Security

[9] H Wang, D Zhang, K G Shin, "Detecing SYN flooding attacks", in Proc IEEE INFOCOM, 2002

[10] Novikov, D.; Yampolskiy, R.V.; Reznik, L, ".Anomaly Detection Based Intrusion Detection", ITNG 2006. April 2006 Page(s):420 – 425

[11] Hixon, R.; Gruenbacher, M., "Evaluation of the Fisher discriminant and chi-square distance metric in network intrusion detection", Region 5 Conference: Annual Technical and Leadership Workshop, 2 April 2004 Pages:119-124

[12] Jun Li; Manikopoulos, C., "Early statistical anomaly intrusion detection of DOS attacks using MIB traffic parameters", Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society 18-20 June 2003 Page(s):53 - 59

[13] K. Wang and S. J. Stolf, "Anomalous Payload-Based Network Intrusion Detection", Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15 - 17, 2004.

[14] B. Trammell, E. Boschi, "Bidirectional Flow Export using IPFIX", Internet-Draft, work in progress, June 2006

[15] J. Yin, G. Zhang, Y. Chen, X. Fan, "Multi-events analysis for anomaly intrusion detection", International Conference on Machine Learning and Cybernetics, 2004

[16] Y. Zhang, Z. Xiong, X. Wang, "Distributed Intrusion Detection Based on Clustering", International Conference on Machine Learning and Cybernetics, 2005

[17] Y. Wang; H. Yang; X. Wang; R. Zhang; "Distributed intrusion detection system based on data fusion method", Fifth World Congress on Intelligent Control and Automation, WCICA 2004.

[18] I. D. Graham, S. F. Donnelly, S. Martin, J. Martens, J. G. Cleary, "Nonintrusive and Accurate Measurement of Unidirectional Delay and Delay Variation on the Internet", INET'98, Geneva, Switzerland, 21-24 July, 1998

[19] N. Duffield, M. Grossglauser, "Trajectory Sampling for Direct Traffic Observation", Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden, August 28 - September 1, 2000

[20] T. Zseby, S. Zander, G. Carle. "Evaluation of Building Blocks for Passive One-way-delay Measurements". Passive and Active Measurement Workshop (PAM), Amsterdam, The Netherlands, April 23-24, 2001.

[21] B. Trammell, E. Boschi, L. Mark, T. Zseby, "An IPFIX-Based File Format", Internet-Draft, work in progress, June 2006

[22] T. Zseby, E. Boschi, N. Brownlee, B. Claise, "IPFIX Applicability", Internet-Draft, work in progress, June 2006

[23] T. Dietz, A. Kobayashi, B. Claise, "Definitions of Managed Objects for IP Flow Information Export", Internet Draft, work in progress, June 19, 2006

[24] G. Muenz, "IPFIX Configuration Data Model", Internet Draft, work in progress, June 2006

[25] A. Fessi, G. Carle, F. Dressler, J. Quittek, C. Kappler, H. Tschofenig, "NSLP for Metering Configuration Signaling", Internet Draft, work in progress, June 26, 2006.

[26] T. Zseby, M. Molina, N. Duffield, S. Niccolini, F. Raspall, "Sampling and Filtering Techniques for IP Packet Selection" Internet Draft, work in progress, July 2005

[27] F. Dressler, C. Sommer, G. Muenz, "IPFIX Aggregation", Internet Draft, work in progress, June 2006

[28] E. Boschi, L. Mark, B. Claise, "Reducing redundancy in IPFIX and PSAMP reports", Internet Draft, work in progress, June 2006

[29] FOKUS IPFIX Implementation, *http://ants.fokus.fraunhofer.de/ipfix/*

# IPFIX/PSAMP: What Future Standards Can Offer to Network Security

FOKUS

Fraunhofer Institute for Open Communication Systems

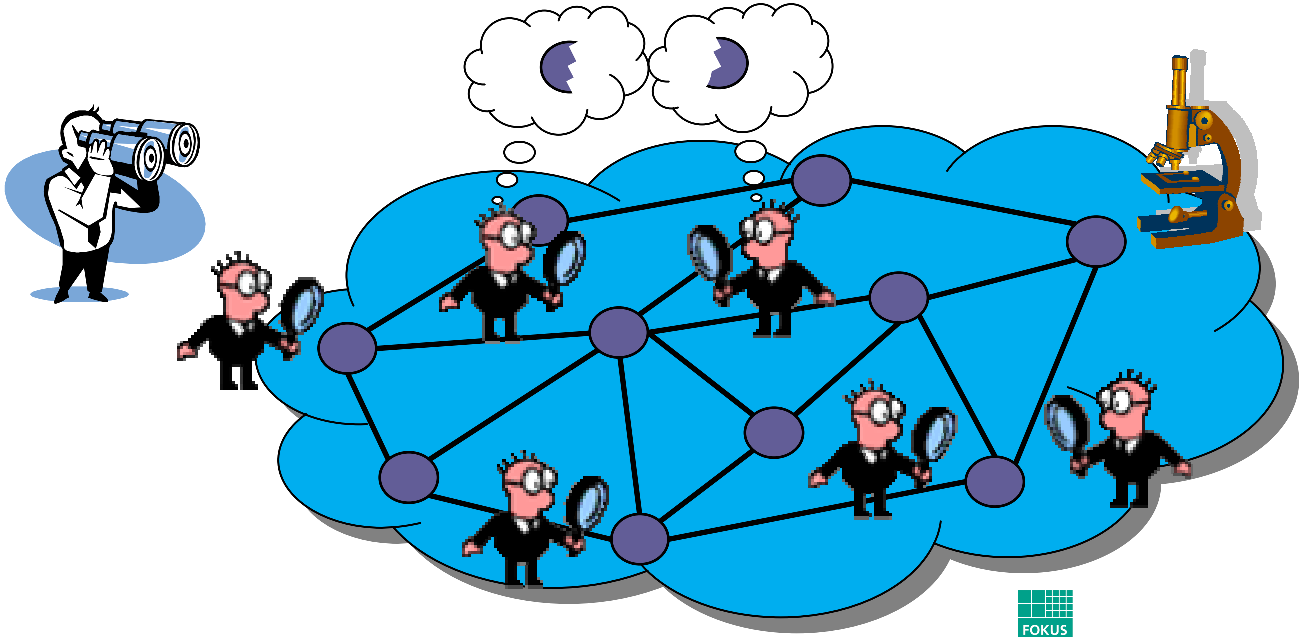
Tanja Zseby, Elisa Boschi, Thomas Hirsch, Lutz Mark

zseby@fokus.fhg.de

# *Measurement Requirements for Network Security*

**Goal: Detect deviations from normal traffic behavior**

- Measurement requirements
  - **Network-wide**: get information from multiple observation points
  - **Flexible**: change viewpoints
  - **Shareable**: provide comparable and shareable results



# Existing Solutions

- Specialized Hardware
  - + Extra resources to capture flow and packet data
  - + Detailed post-incident analysis possible
  - - Huge amount of measurement data ➔ high analysis effort
  - - Network installation required ➔ operators distrust new devices
  - - High costs ➔ prevent network-wide deployment
- SNMP
  - Useful, but too coarse grained information
- Proprietary measurement tools
  - > 400 different tools (academia, research, operators, etc.) ➔ www.ist-mome.org
  - - Require additional devices ➔ prevent network-wide deployment
  - - Different input/output formats ➔ hard to share and compare
- Cisco NetFlow
  - + Integrated in routers ➔ network-wide deployment
  - - Fixed flow definition, no packet data ➔ limited flexibility
  - - High resource consumption ➔ Router performance degradation
  - - UDP transport ➔ potential data loss, no congestion control

# IETF Standardization Efforts: IPFIX

**IPFIX - IP Flow Information EXport**

- Protocol for flow information export
  - Exports flow data from routers and probes (IPv4, IPv6)
  - Works on top of UDP, TCP or SCTP
  - Similar to Cisco NetFlow but much more flexible
- Upcoming IETF Standard
  - Active IETF working group
  - Protocol draft in last call
  - First Implementations exist
- Target Applications [RFC3917]
  - Usage-based Accounting
  - Traffic Profiling
  - Traffic Engineering
  - **Attack/Intrusion Detection** ←
  - QoS Monitoring

# IPFIX Details

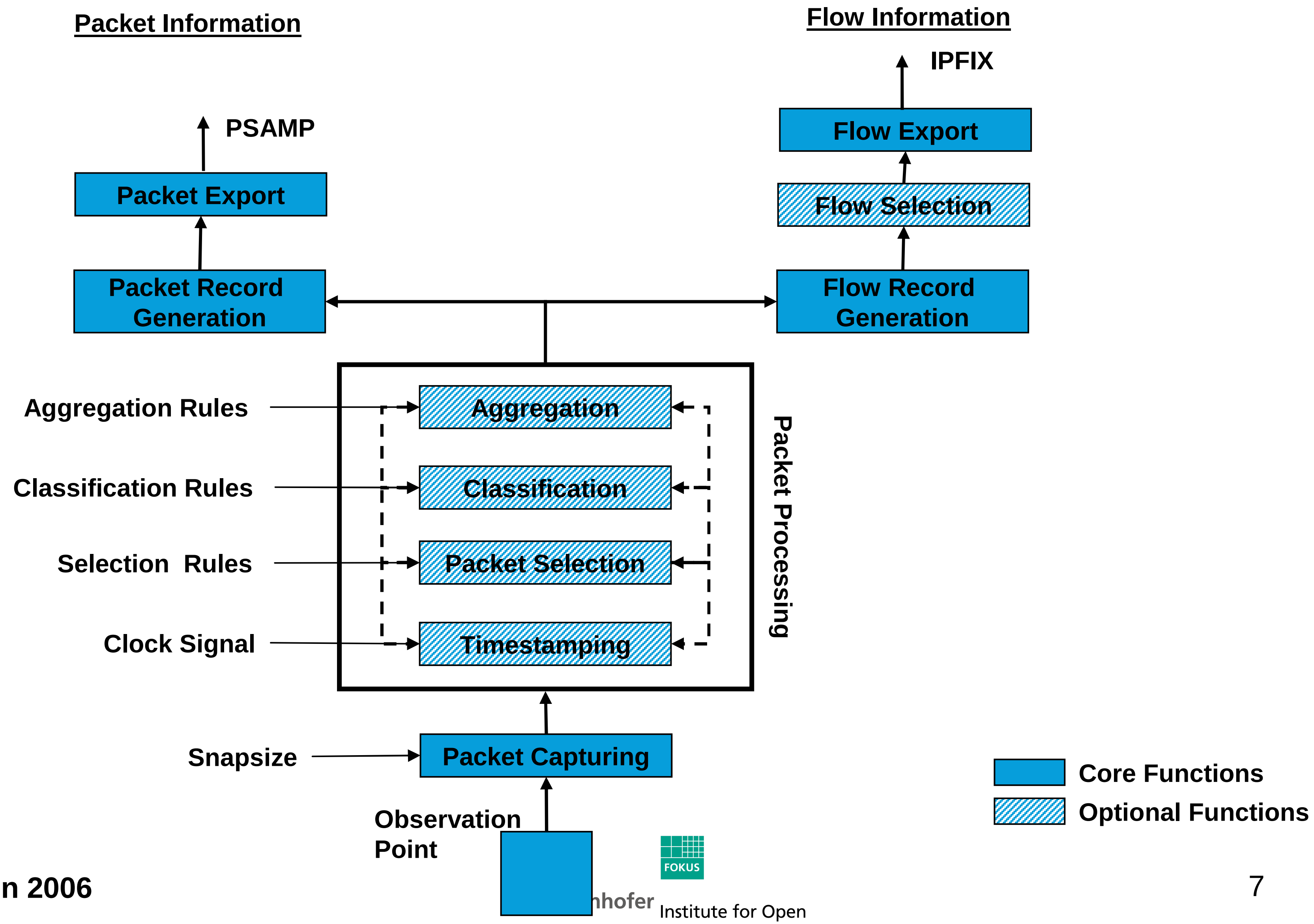
- Template-based approach
  - **Template Records**: define structure of Data Records
  - **Data Records**: contain parameter values
  - **Option Template Records**: provide additional information for Collectors
- Push-Model
  - Flow records pushed from exporter to collector
  - Trigger not defined in IPFIX
    - Measurement configuration out of scope
    - Flow termination criteria currently used, but others possible
- Information Elements (IEs)
  - Base sets of IEs defined in IPFIX-INFO, PSAMP-INFO
  - Attributes that can appear in IPFIX records
  - Vendor-specific IEs can be defined

# IETF Standardization Efforts: PSAMP

**PSAMP - Packet Sampling**

- Exporting packet information with IPFIX
  - IEs for reporting packet header and payload
  - PSAMP IEs defined in draft-ietf-psamp-info-04.txt
  - PSAMP Framework in draft-ietf-psamp-framework-10.txt
- Packet selection methods
  - Filtering: deterministic selection based on packet content
  - Sampling: random or deterministic selection
  - PSAMP Schemes in draft-ietf-psamp-sample-tech-07.txt

# IPFIX/PSAMP Measurement Model



# What IPFIX/PSAMP can offer to NW Security

- Network-wide measurements
  - Measurement results from routers
  - No extra devices required
  - Different transport protocols (e.g. for congestion control)
- Highly flexible measurement definition
  - Arbitrary packet and flow information, highly flexible flow definition
  - Data selection techniques
  - Extensible information model
- Comparable and shareable data
  - Standardized data format
  - Different aggregation levels and sampling to enhance privacy
  - Secure data exchange (e.g. among domains)

**IPFIX applicability statement: draft-ietf-ipfix-as-10.txt**

# Reporting Flow Statistics with IPFIX

**That’s what IPFIX was designed for!**

- Very flexible flow definition
  - Any set of packets with “common properties” defined by flow keys
    - Packet header fields (e.g. destination IP address)
    - Packet properties (e.g. number of MPLS labels)
    - Packet treatment (e.g. output IF)
  - Information elements usable as flow keys defined in IPFIX-INFO
    - All IPv4 header fields (except checksum)
    - Main IPv6 header fields (addresses, next header, flow label, etc.)
    - Main transport header fields (UDP, TCP ports, sequence num., ICMP types)
    - Some sub IP header fields (MAC addresses, MPLS labels, etc.)
  - Flow termination criteria (currently used)
    - Idle timeout (no activity)
    - Active timeout (active, but max lifetime expired)
    - End of Flow detected (e.g. TCP FIN observed)
    - Forced end (external event, e.g. shut down of the Metering Process)
    - Cache full (lack of resources)

# Reporting Flow Statistics with IPFIX

- Variety of information elements to report flow characteristics
  - Counters (e.g. bytes, packets, delta and total counters)
  - Timestamps (flow start, end, duration)
  - Statistics (min/max pktlength, min/max TTL, TCP flags, options)
  - Others (e.g. flow end reason)
- Per-flow TCP Flag counters
  - Recently introduced in IPFIX-INFO
  - E.g. tcpSynTotalCount, tcpFinTotalCount
  - Useful for detection of claim&hold attacks (e.g. SYN flood)

# Bi-directional Flows

- Reporting both directions of a communication is useful for NW security
  - Connection status: incomplete connections can indicate attacks
  - Check request/response pairs (DNS, etc.)
- BUT: IPFIX currently reports each direction as separate flows ➔ How to report bi-directional flows?
- With standard IPFIX (without extensions)
  - Approach 1: Two records with record adjacency
    - unidirectional flow records adjacent to each other, collector reassembles
    - + extremely simple
    - - maintaining right or

SEE BI-FLOW TALK

    - set of properties
    - Flow records (for each direction) carry individual uniflow properties (references keys by commonPropertiesID)
    - + more efficient
    - - additional resources for managing commonPropertiesID (at exporting and collecting process)
    - - three records required (instead of two)
- With IPFIX extension
  - Definition of new IEs for reverse direction
  - Re-use existing IEs and use special vendor ID to separate forward and backward direction
- Approaches currently discussed in draft-trammell-ipfix-biflow-02.txt ➔ best method will be selected

# Packet Captures

- IPFIX: only header information
  - Define each packet as separate flow
  - IP, transport header, and some sub IP information per packet
  - Flow keys reported for each packet → inefficient
- IPFIX improved export
  - Sharing flow key information among data records
  - → Methods discussed in reduced redundancy draft
- With PSAMP
  - Header: ipHeaderPacketSection
  - Payload: ipPayloadPacketSection
  - Sub IP: dataLinkFrameSection, mplsLabelStackSection, etc.
- Data reduction
  - Aggregation of flows
  - Packet selection methods ← **PSAMP**

# PSAMP Packet Selection Schemes

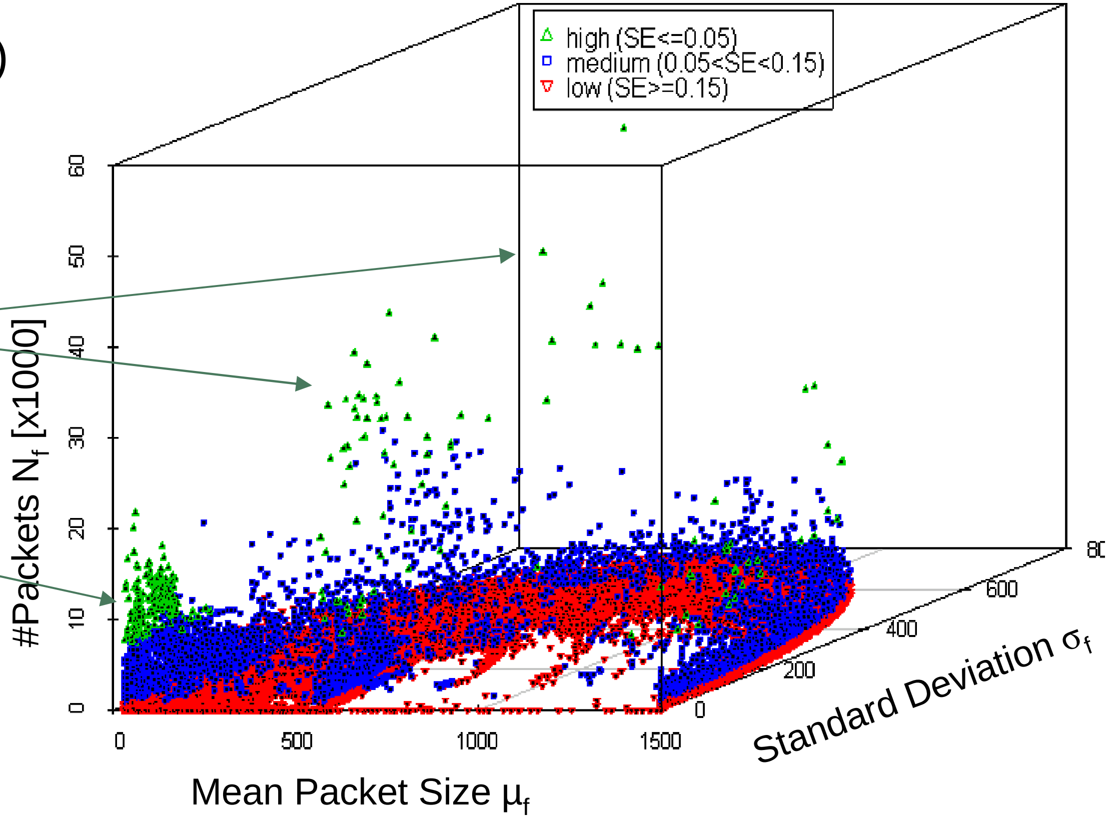
- PSAMP offers basic packet selection techniques
  - Filtering: deterministic selection based on packet content
    - Mask/match filter
    - Hash-based selection
    - Router state filter
  - Sampling: random or deterministic selection
    - Systematic count-based
    - Systematic time-based
    - Random n-out-of-N
    - Random uniform probabilistic
    - Random non-uniform probabilistic
    - Random non-uniform flow-state
- Packet selection possible at different points in measurement process
- Concatenation of selectors possible (e.g. for stratified sampling)
- Flow sampling
  - Allowed in IPFIX architecture
  - Currently not defined in PSAMP

# Sampling Example: Achievable Accuracy

**Example: Flow volume estimation**

- PSAMP random n-out-of-N (5%)
- Sampling ***before*** classification

**C/C/RN, Case B, f=5%, S24D24**



***High accuracy for large flows***

***High accuracy for flows with small variation***

$$SE_{rel} = \frac{\sqrt{\frac{N \cdot N_f}{n} \cdot \left(\sigma_{x_f}^2 + \mu_{x_f}^2\right) - \frac{N_f^2}{N} \cdot \mu_{x_f}^2}}{N_f \cdot \mu_{x_f}}$$

**➔ Large flows detectable with very small effort**

**➔ MORE ON SAMPLING IN PANEL**

# IPFIX Configuration

- Past: Configuration was out of scope for IPFIX
  - WG wanted to concentrate on protocol spec
  - Proprietary CLI configuration of IPFIX processes always possible
- Now: Several Proposals for IPFIX configuration
  - IPFIX MIB (draft-dietz-ipfix-mib-00.txt)
    - Monitoring IPFIX exporters and collectors (configuration, statistics)
    - Potentially configuration of IPFIX exporters and collectors
  - IPFIX XML configuration (draft-muenz-ipfix-configuration-00.txt)
    - Data model for configuration parameters of IPFIX devices
    - Configuration by Netconf, SOAP, etc.
  - NSIS proposal (draft-dressler-nsis-metering-nslp-04.txt)
    - Path-coupled dynamic configuration of Metering Entities
    - Metering NSIS Signaling Layer Protocol NSLP (M-NSLP),
    - Cooperation between NSIS and IPFIX required

# Storage of Data

- Standardized format for storing IPFIX data
  - Post-incident analysis (forensics, research)
  - Sharing information (e.g. among providers)
  - Provide training data (traces with “normal” behavior)
- IPFIX file format draft (draft-trammell-ipfix-file-01.txt)
  - Collects Requirements
    - Extensibility (multiple record types, new fields, etc.)
    - Self-Description (interpretation without additional knowledge)
    - Data Integrity and Error Correction
    - Authentication and Confidentiality
    - Indexing and Searching
    - Anonymization
- Goal: propose an IPFIX file format
  - Evaluation of existing solutions (ARGUS, SiLK, etc.)
  - Collection of requirements

# Support in Routers

- IPFIX (Flow Export)
  - First Implementations exist
  - Cisco plans IPFIX compliance
- Packet Export
  - Resource limitation on routers prevent full packet export
  - Packet export from sampled data possible
  - Tradeoff between reported amount of information (#packets, snapsize) and required resources
- Sampling Methods
  - Cisco: random 1-in-K, systematic sampling
  - Conformance to PSAMP if one PSAMP scheme is supported
  - No information about support for further schemes

# Conclusion

- IPFIX/PSAMP
  - Protocol to export flow and packet information
  - Upcoming standard
  - Can integrate data selection methods
- Provides measurement results
  - Network-wide
  - Flexible
  - Shareable

➔Powerful standards for network security

FOKUS Open Source IPFIX library available at:

**http://ants.fokus.fraunhofer.de/libipfix/**

# Thank you for your attention!

FOKUS

Fraunhofer Institute for Open Communication Systems

# The Past and Future of Flow Analysis

John McHugh
Canada Research Chair in Privacy and Security
Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada
My-last-name at cs dot dal dot ca

# Greetings from Canada

# The anomaly of keynotes

- All of a sudden, I’m being asked to give keynotes at various conferences and workshops.
- I suspect that it has something to do with advancing age, approaching senility and the desire of the organizers to give the jet lagged participants an extra hour of sleep on the first day.
- Nonetheless, it gives me a chance to express opinions without having to back them up with facts that have to pass muster with reviewers.

# Aphorisms

- (1) Pay attention to details. (2) Don't make stuff up.

  - Roy Maxion
- “She got the goldmine, I got the shaft”

  - Song by Jerry Reed
- Your mileage may vary

  - From the standard EPA disclamer

# Outline - 4 Themes in search of an author

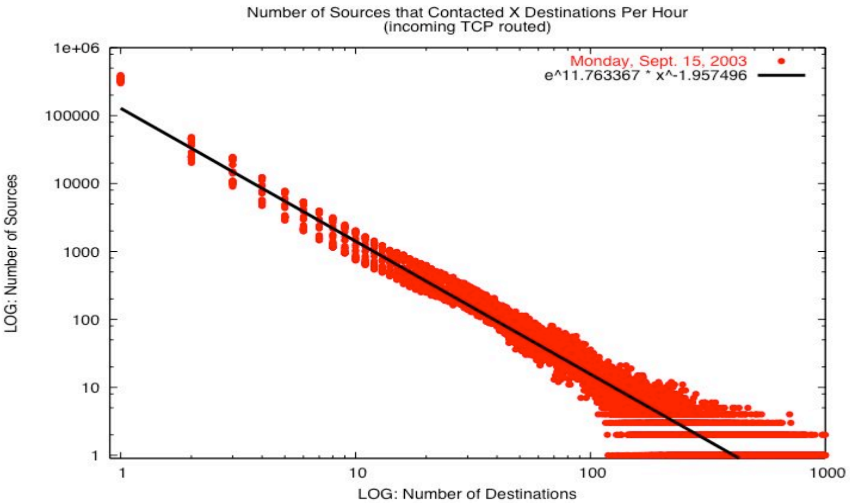
- Parkinson’s law
- Is it good for anything else?
- “Look homeward angel”
- Cast your net broadly

# Parkinson's Law

- Parkinson's Law
  - Prof. Cyril Northcote Parkinson , 1958
  - **“WORK EXPANDS SO AS TO FILL THE TIME AVAILABLE FOR ITS COMPLETION”**
- The corollary is obvious, the answers are not.
  - Buy stock in your favorite disk vendor
  - Figure out how to break the law
  - Lets look a bit at the later

# Distance and precision

- Look at the contact line distribution
- Are there regions where we can usefully abstract away the details?



# Infrequent  and super frequent contact

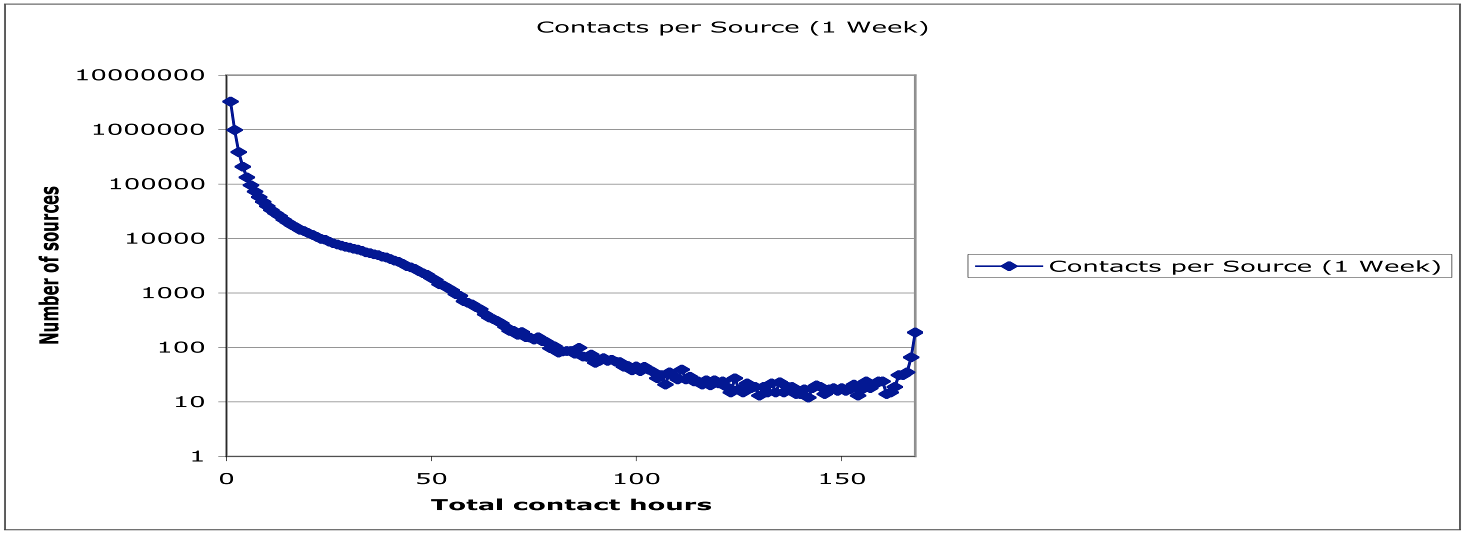
- We see that there are a lot of sources that appear once per hour.
- There are a handfull that appear a lot.
- If we could represent these more efficiently, lots of space could be saved
- The underlying assumption is that there is not a single mechanism at work here, but the results of multiple mechanisms and they can be treated separately.

# Singletons

- We haven’t really looked too closely at these since there are millions per hour, but.
  - The majority appear to form a small number of groups based on protocol, ports, flag combinations, etc.
  - I suspect that most are addressed to unoccupied addresses
  - Looking back at data that is months or years old, do you really care what the individual target was or when in the hour the flow occurred?
  - If not, some lossy compressions are obvious

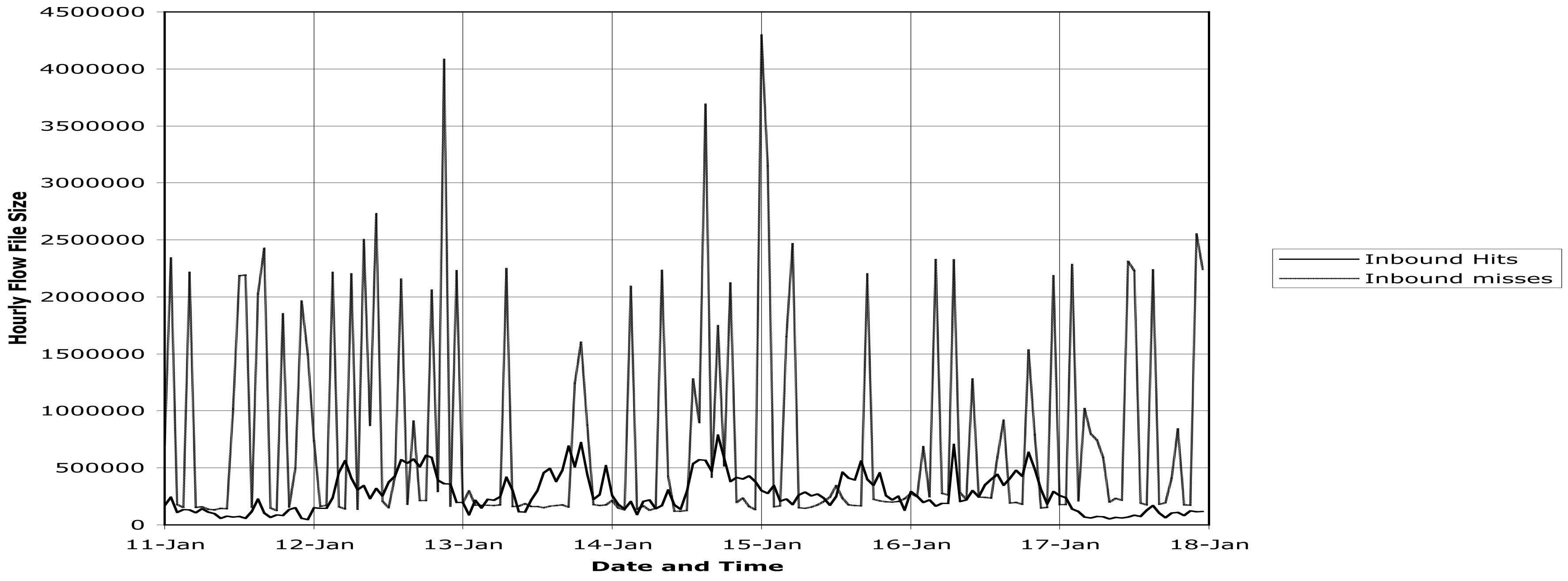
# More singletons

- Suppose we look at the one/hours over a week
  - Again, the majority are never seen again
  - This reinforces the earlier discussion



# Hyperactive sources

- Most of these are high volume scanners.
- Most of the scans do not find targets



# Typical scan

- Looking at a single spike, one often finds things like:

```
          sIP|dPort|pro|packets|   bytes|   flags|
 AA.BB.x.y|    80|  6 |        2|      88| S      |   96842
 AA.BB.x.y|    80|  6 |        1|      44| S      |   20233
 AAA.BBB.0.0/16 : 51744 hosts in 228 /24s and 1785 /27s.
```

- Again, there are obvious compressions that trade precision for volume
- Note that backscatter is another source of hyperactivity, but similar processing applies

# Other tricks

- I suspect that treating low levels of activity and high levels of activity will reduce the volume of the remaining data by 75 to 95 % (YMMV).
- At that point, further processing of the main stream might be useful
  - Session reconstruction comes to mind
  - Aggregation of similar sessions
  - Etc.

# When is not a flow, a flow?

- Reduced to its essentials, the SiLK paradigm is a way of thinking about intermingled time series that are glued together by common factors.
  - Think of the paradigm as a way of reasoning about large aggregations of event data.
  - Convert the data into pseudo flow records and have at it.
  - The essentials are some sort of anchors (IP addresses), possibly some volumes (1 is a perfectly good default), and maybe some other stuff (tags, flags, ports, etc.)

# Packets are obvious

- For a DARPA project I worked on at CERT, it was extremely useful to apply SiLK style filtering to packet data.
- My vision was “rwpfilter”, a program that would bring the power of rwfilter to packet data and extract pass/fail files containing both packets and degenerate (1/pkt) flows.  For packets,it would be possible to filter payload with regular expressions
- A prototype was built, but it has been an uphill battle to bring this into the main stream.  I have a MS student who is going to try, and the latest rwptoflow starts to be a building block.

# Other issues

- DHCP
  - Especially in wireless networks
- NAT
- Sensor identification issues, etc.
- Inside to outside distinction and grouping
  - In -> Out
  - Out -> In
  - In -> In
  - Out -> Out ???

# Crawdad / predict / etc.

- Dartmouth has a repository of 4 months of pkt headers from their wireless network
  - 160GB compressed / 18 sensors
  - Anonymized (badly)
  - 17000 MAC addresses also badly anonymized
- Converted to hourly flows in early 2006 and used in my course.
- Students were able to identify a number of interesting things including several worm infestations, but …
- DHS may bring the predict archive on line soon
  - This is a potential source of data, but … YMMV

# Other sources

- With a text to flow program and a few text processing scripts, the possibilities are interesting.
  - Log data from firewalls, IDS, etc.
  - Pilot study involving data from a large managed services company looked promising and could produce “top N” reports easily.

# Generalize, Generalize, Generalize

- Extend SiLK tools for filtering, set and bag production, etc. to all scalar fields in the archive format.
  - Sets of sensors make sense
  - So do sets of flags, etc.
  - In a strange way, so do sets of times or durations.
  - I experimented with Bloom filters for detecting connection level service activity (SIP, DIP, service)

# Plagiarize, Plagiarize, Plagiarize,

**Only, please, call it research**

- Support set and bag structures, along with hash trees, Bloom filters, perfect hashes, etc. at a library level for specialized analyses.
- The dynamic library concept allows a lot of creative use of the concepts. Make it easy on the researcher analyst to use the bits and pieces of the tools
  - Internals documentation?
  - Builders guide?
  - Skeleton programs?

# Know thy network!

- Much of the SiLK work at CERT has been focused on border data from a large network.
- There is growing evidence that continuous monitoring at the small enterprise level is useful for all sorts of things, including security, but also for provisioning.
- You will see two papers later in the workshop on one such effort performed by Ron McLeod.
- Even if there are no large scale compromises, such monitoring often provides interesting insights
    - Why do both of the cases in which we monitored a cable modem show 95% ARP?

# More Data, More Data …

- The sources represented at this meeting are largely big institutional networks.
- Several people have recently suggested a co-op approach to data collection and sharing.
  - Eurecom has a honeypot co-op.  Contribute by running their honeypot system and you get access to all the reported data.
  - Anyone for a flow co-op from your home DSL / cable modem drop?
  - There are a number of collectors and ideas for better ones …

# Thanks

- Tim for inviting me.
- SLK for seizing the moment
- Tom Longstaff for creating the environment that made it all possible
- Mike for keeping the spirit alive in trying times
- All the developers for their support, understanding and willingness to take suggestions.

- QUESTIONS …

# If it ain’t broke - Don’t IPFIX it.

# Attribution and Aggregation of Network Flows for Security Analysis

Annarita Giani, Ian Gregorio De Souza, Vincent Berk, George Cybenko
*Thayer School of Engineering*
*Dartmouth College*
*Hanover, NH*
{*agiani, idesouza, vberk, gvc*}*@dartmouth.edu*

## Abstract

This paper describes a network flow analyzer that is capable of attribution and aggregation of different flows into single activity events for the purposes of identifying suspicious and illegitimate behaviors. Flows are correlated with security events using the Process Query System (PQS) infrastructure. We show results from initial experiments and describe plans for extending the effort. The correlation of networks flows with security events appears to have high potential for aggregating disparate network and host activity and for classifying network activity as either benign or suspicious.

## 1 Introduction

A flow sensor emits observations as detailed flows. The system generates a very large continuous quantity of data that cannot be processed by humans practically. A skilled security analyst that tries to analyze the flow characteristics may recognize an illegitimate connection but is likely to miss some malicious traffic. The fact that a flow is unjustifiable can be related to the flow breaking some criterion described in a policy file or to the fact that intrusion detection systems (IDS) or other sensors had evidence classifying the flow as illegal. For example, if we notice that a flow contains an FTP session and the initiator of the connection is a host that triggered a Snort sensor, the flow must be categorized as malicious. A system administrator might miss this correlation and report the connection as legitimate.

We have designed and implemented an approach to correlating network flows with security events, such as those generated by IDS, for the purposes of aggregating and attributing flows. This capability achieves data compression and provides insights to the analyst about whether flows are benign or suspicious. Our temporal and multi-sensor correlation system is based on a Process Query System which is designed to correlate different data based on the process models that it runs. Models serve to correlate data coming from flow anomaly detectors and other security tools.

Process Query Systems (PQS) [2, 7] are software systems that allow users to interact with multiple data sources, such as traditional databases and real-time sensor feeds, in a new and powerful way. In traditional databases, users specify queries expressed as constraints on the field values of records stored in a database or data recorded by sensors, as allowed by SQL and its variants for streaming data. By contrast, PQS allows users to define processes and to make queries against databases and real-time sensor data feeds by submitting those process definitions. A PQS parses the process description and performs sequences of queries against available data resources, searching for evidence that instances of the specified process or processes exist.

The strength of the system relies on the fact that the same PQS engine can be easily adapted to different processes. PQS has been successfully applied to ground vehicle tracking [6], social networks [4], and plume source detection [10]. For each of the preceding applications, the observations gathered are, respectively: positions as recorded by sensors, evidence of particular activity in the social network (initiator of a conversation, broker, etc.) and concentrations of a particular gas at a fixed location.

PQS is also a very powerful and efficient tool to perform network security monitoring. In this framework the system collects data from many different sensors and implements multilevel models that evaluate the data. As a result it returns conclusions that can be very specific or more general depending on which PQS tier is used. Previous implementations of PQS models [8, 9, 1] have detected malicious hacker behaviors, insider threat behaviors, network failures, worms, viruses and covert channels. In this implementation of the system we want to disambiguate between benign and malicious behavior in terms of flows. We want to develop a framework for using flow attribution and flow aggregation as part of security analysis. *Flow attribution* consists of detecting logs that can explain a flow. The final goal is attributing the flow to a person but intermediate steps are a required part of the attribution process. *Flow aggregation* means recognizing that different flows, apparently totally unrelated, nevertheless belong to the same broader event. Single flows are defined as components and groups of related flows are defined as events. Examples are an FTP connection followed by data transfer or surfing the web and connecting to different web pages following links present in the pages.

Flow attribution has been investigated mostly with the purpose of enhancing quality of service [5, 3] while the present work is devoted to security uses of network flows.

The paper is organized in the following way. Section 2 describes the flow sensor together with a description of the network in which it was deployed. Section 3 presents some of the other sensors used in our analysis. Section 4 shows results of an experiment that we ran and anticipates other experiments. Section 5 gives a brief overview about how to measure improvements in this framework.

## 2 Flow Sensor Description

The flow sensor that we built was deployed on an unsecured production network [1]. Although the network is small compared to an actual enterprise-class network, it features a wide range of systems and servers that are typical of larger organizations. Behind the firewall there are 64 addresses available (.192/26) which are split between a *Workstations* and a *Demilitarized Zone* server network, both (/27). The uplink connects directly to the internet and all addresses are globally routable. The Workstation network features several Windows XP clients, Linux 2.4 and 2.6 systems, several Solaris workstations, and a Solaris 9 server to which multiple thin-clients are connected. All these systems are used daily by researchers and students, and so the traffic on this network is typical for a normal operating organization where users browse the web, print documents, and download files during business hours.

Our flow sensor is based on the libpcap interface for packet capturing. Traffic flows was collected for a period of two months. Packets with the same source IP address, destination IP address, source port and destination port and protocol were aggregate in a single flow which remain active until no other packets with the same characteristic arrive for a period of 5 minute.. Each flow is characterized by fields that are used to infer quantities given as observations to the PQS models. These fields are:

1. Timestamp of the first packet.
2. Timestamp of the last packet.
3. Number of packets from the source to the destination
4. Number of packets from the source to the destination
5. Number of bytes from the source to the destination
6. Number of bytes from the destination to the source
7. The IP address of the source machine
8. The IP address of the destination machine
9. The protocol
10. The source port
11. The destination port
12. An array containing the delays, in microseconds, between two consecutive packets
13. An array containing the number of bytes in each packets.

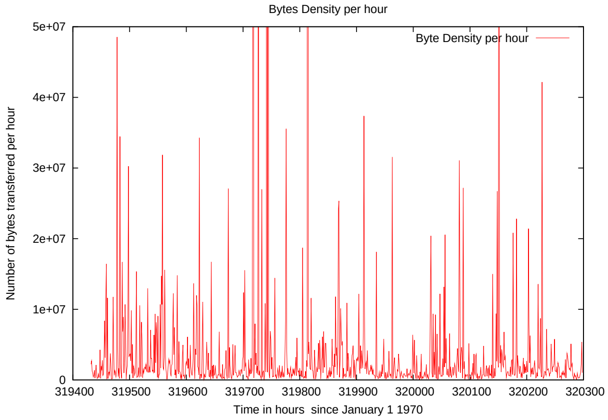


Figure 1: Bytes Density per hour from June 09 2006 04:00:00 to July 16 16:00:00

These quantities allow us to infer interesting statistics on the network traffic on our network. Figure 1 and Figure 2 show the density of flows and bytes transferred per hour respectively. We notice for example that the number of flows per hour is mostly between 50 and 100 with some sporadic spikes.

The delays between packets and the distribution of bytes in each flow embed information that can be used to investigate situations of covert channel embedded in inter-packets delays and also provide statistics on the distribution of load in each flow which will be relevant to the development of higher-level security tools.

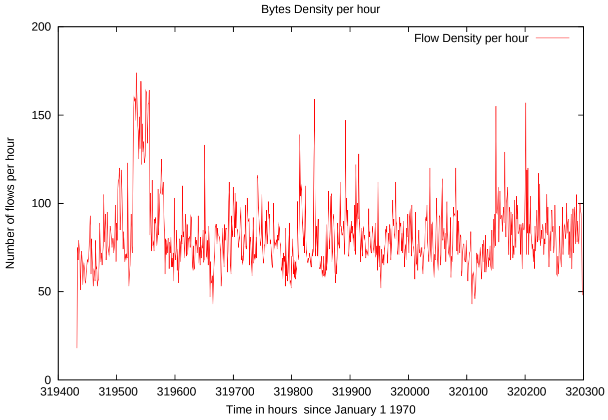


Figure 2: Flow Density per hour from June 09 2006 04:00:00 to July 16 16:00:00

## 3 Host and network sensors

PQSnet is our implementation of a PQS to the domain of computer security. Since the beginning of the PQSnet project three years ago, we have used a number of sensors based on common host and network security applications.

The *Snort* and *Dragon sensors* are both signature-matching Intrusion Detection Systems (IDS). The sensor's observations give information on source and destination IPs, ports, and which rule or signature was violated.

The *Web log sensor* reports information from Apache, IIS and SSL error logs. Observations from this sensor contain information about suspicious URLs that have succeeded or failed, as well as web server errors.

The *Samhain sensor* reports alerts from the Samhain host file integrity monitor. Samhain frequently checks critical system files on a host for additions, modifications and deletions. Any changes are immediately logged locally or reported to a remote log server. Samhain sensor observations include timestamps of changes, filenames, violation type, and changes in the system kernel. Observations from this sensor allows correlation of network and host based activity.

The *NetFlow sensor* classifies a communication between two or more machines in terms of volume of data transfer, length of the transmission and number of packets involved. The sensor monitors network packets and aggregates them in groups of the same source IP, destination IP, source port, and destination port. As packets in the same flow arrive, the number of bytes of new packets are recorded. The flow is considered closed if transmission stops for the duration of a specified time threshold. At that point, the number of packets and total duration of the flow is computed. These quantities are then classified as shown in Table 1. This is the first version of network flow sensor that we built. We still use it since it is useful to get observations that feed models like *large upload*, *low and slow download*, that are instantiated according to the volume, number of packets, duration and balance. Percentage is the actual percentage above 50 of data movement in the direction of the balance field.

| Volume | Packets | Duration | Balance | % |
|---|---|---|---|---|
| 0: 0<br>1: 1-128b Tiny<br>2: 128b-1Kb Small<br>3: 1Kb-100Kb Med.<br>4: $>$ 100Kb large | 1: one packet<br>2: two packets<br>3: 3-9<br>4: 10-99<br>5: 100-999<br>6: $>$ 1000 | 0: $<$ 1 s Zero<br>1: 1-10 s Very Small<br>2: 10-100 s Small<br>3: 100-1000 s Medium<br>4: 1000-10000 s Large<br>5: 10000-100000 s Very Large<br>6: $>$ 100000 s Very Very Large | 1: IN<br>2: OUT<br>3: PAR | N |

Table 1: NetFlow sensor classifications

## 4 Investigated scenarios

## 4.1 Flow attribution

### 4.1.1 Packets in a flow triggered IDS alerts

The PQS instantiates models based on observations coming from the flow and the Snort sensors. An example of a track formed is shown in Table 2. The columns represent the timestamp, the sensor type ($F$ = Flow Sensor and $S$ = Snort), the IP address of the initiator of the communication, the destination IP address, and the protocol respectively. This particular track shows that a single flow between two hosts triggered many Snort alerts. The numbers next to the Snort sensor type represent the particular Snort rule that was violated. Snort rule 1560 (*WEB-MISC /doc/access*) generates an alert when an attempt is made to exploit a known vulnerability on a web server or a web application. Snort rule 1852 (*WEB-MISC robots.txt access Summary*) generates an alert when an attempt is made to access the 'robots.txt' file directly. Also we must notice the starting and ending timestamp of the flow. As we can see the flow ends but the model keeps correlating the flow with snort alerts generated after the flow finished.

## 4.2 Flow aggregation

The number of flows collected over a network can be very high and belong to disconnected activities like chatting or retrieving a web page. But flows apparently unrelated to one another might still belong to a single user

| Timestamp | Sensor | src IP | dst IP | Proto |
|---|---|---|---|---|
| Jul 09 16:28:32 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 16:29:35 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 16:44:44 | S1560 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 18:26:08 | S1560 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 21:05:03 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 22:31:08 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 22:31:08 | S1560 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 02:45:19 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 02:45:23 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 09:21:15 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 14:33:43 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 17:54:54 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 22:07:02 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 01:38:09 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 04:05:54 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 04:20:00 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 04:20:00 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 11:07:12 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 11:56:12 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 17:16:59 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| S Jul 10 02:30:27<br>E Jul 10 23:55:56 | F | 65.54.188.140 | 208.253.154.195 | TCP |

Table 2: A sample track of correlated IDS and Flow events

event or action. Our goal is to correlate all these flows to a single user activity.

### 4.2.1 Surfing the web

The full retrieval of a web page is a classical example of this situation. The first flow represents a DNS query to retrieve the IP address of the web server. An HTTP query to the web server generates a second flow. At this point many other flows may be generated (for example, as images download).

Another more challenging example is the following. Users often start connecting to a web page, and then follow links present in the page they visit. Since each page might have different flow characteristics this activity results in many different flows. But it is reasonable to think about these flows as part of the same event. Time stamps can be used to discriminate different instantiations of the same event. The challenge is to identify unrelated flows as part of a single event.

Building models for web browsing aggregation requires retrieving flows with destination port 80, parsing the web pages and following the links. If an IP address or URL linked by the web page is present in one of the flows it might be that the two requests are connected.

## 5 Improvements metrics

While developing a framework for flow tracking it is important to measure the success of our system. We must therefore find ways to quantify progress. One way of measuring improvement in our method is checking the size of the group of actions to which a given observable or flow may be attributed. If the size as well as the ambiguity reduces the methodology becomes more and more precise. Each attribution or aggregation must be supported by a level of confidence quantified with a number indicating how certain we are of the conclusions. Also, we can improve the system by increasing the number and type of observations that it can aggregate into a set of basic components.

## 6 Conclusion and Future Work

Process Query Systems are powerful tools to perform sensor observation correlation. We applied the system to correlate flow observation with other IDS observations successfully. We plan to run experiments to perform flow aggregation in order to identify flows belonging to a single user event. We believe that this would lead to significant improvements in security analysis.

## 7 Acknowledgments

Supported under ARDA P2INGS Award No. F30602-03-C-0248. Points of view in this document are those of the author(s) and do not necessarily represent the official position of ARDA.

## References

[1] V. Berk and N. Fox. Process Query Systems for Network Security Monitoring. *Proceedings of the SPIE*, 5778, April 2005.

[2] V. H. Berk, W. W. Chung, V. Crespi, G. Cybenko, R. Gray, D. Hernando, G. Jiang, H. Li, and Y. Sheng. Process query systems for surveillance and awareness. *Proceedings of Systemics, Cybernetics and Informatics (SCI2003*, July 2003.

[3] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt. Flow Aggregation for Enhanced TCP over wide-area wireless. pages 1754–1764, 2003.

[4] W. Chung, R. Savell, J. P. Schtt, and G. V. Cybenko. Identifying and Tracking Dynamic Processes in Social Networks. *Proceedings of the SPIE*, 6201, April 2006.

[5] J. A. Cobb. Preserving Quality of Service Guarantees in spite of Flow Aggregation. *IEEE/ACM Trans. Netw.*, 10(1):43–53, 2002.

[6] V. Crespi, W. Chung, and A. B. Jordan. Decentralized Sensing and Tracking for UAV Scheduling. *Proceedings of the SPIE*, 5403, April 2004.

[7] G. Cybenko, V. H. Berk, V. Crespi, R. S. Gray, and G. Jiang. An Overview of Process Query Systems. *Proceedings of the SPIE*, 5403, April 2003.

[8] I. G. de Souza, V. H. Berk, A. Giani, G. Bakos, M. Bates, G. Cybenko, and D. Madory. Detection of Complex Cyber Attacks. *Proceedings of the SPIE*, 6201, April 2006.

[9] A. Giani, V. Berk, and G. Cybenko. Covert Channel Detection Using Process Query Systems. *FloCon*, 2005.

[10] G. Nofsinger and G. Cybenko. Distributed Chemical Plume Process Detection. *IEEE MILCOM*, 2005.

# Attribution and Aggregation of Network Flows for Security Analysis

Annarita Giani
Ian De Souza
Vincent Berk
George Cybenko

Institute for Security Technology Studies
Thayer School of Engineering
Dartmouth College
Hanover, NH

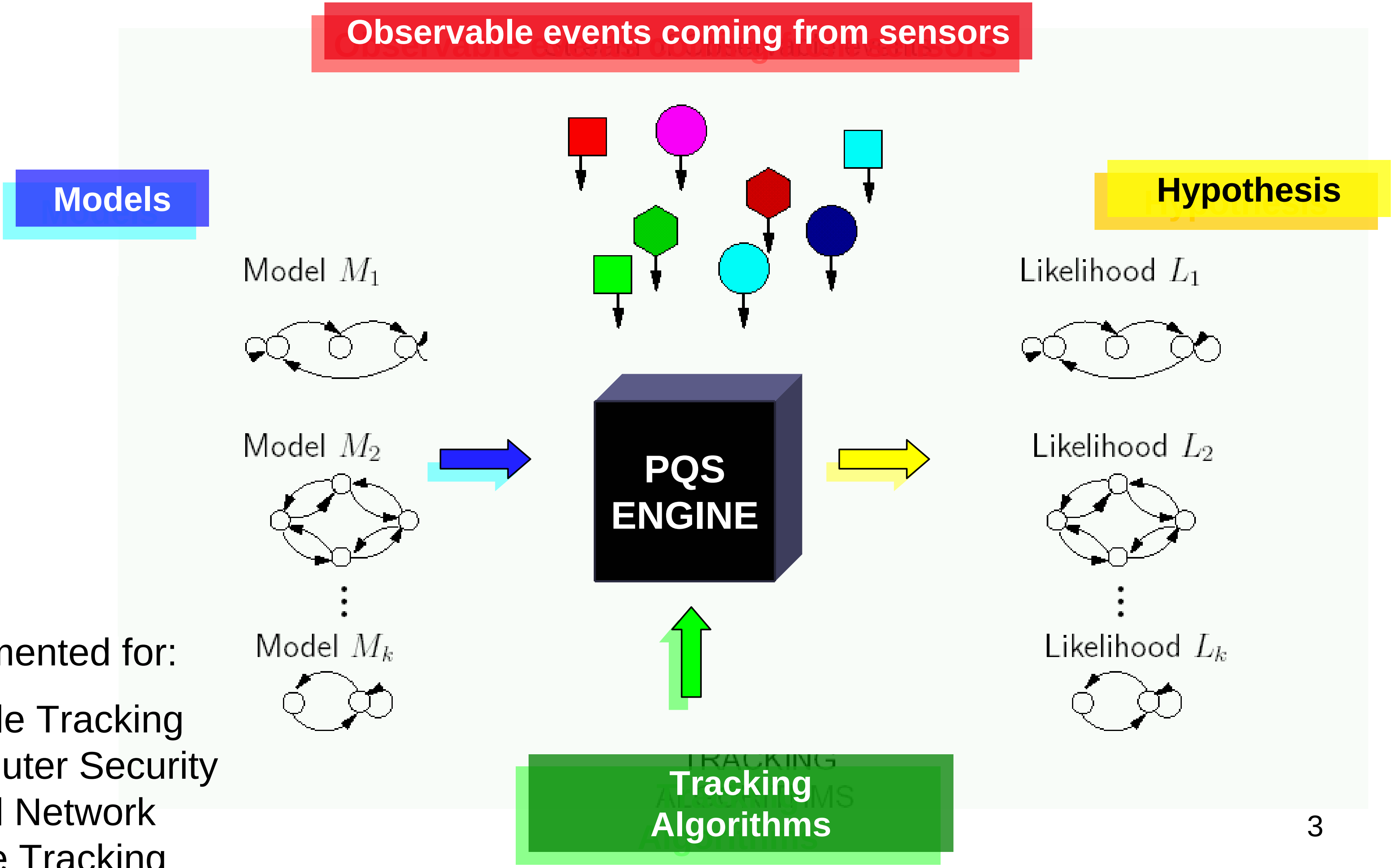
FloCon 2006, Portland, OR

# Why flow data

The context in which we are interested in flow analysis is the following.

- We believe that **automated correlation** is hard to do.
- The world consists of **processes** so our approach to correlation is process-based..
- Introduction, in 2003, of generic process-based correlation engine concept and implementation, **Process Query System** (PQS).
    - Integration of multiple existing and new sensor types and attacks models
    - Flow aggregation and correlations between flow data with security events
    - Implementation of a **PQS based process detection for Cyber Situational Awareness.**
    - **Need for flow data**.

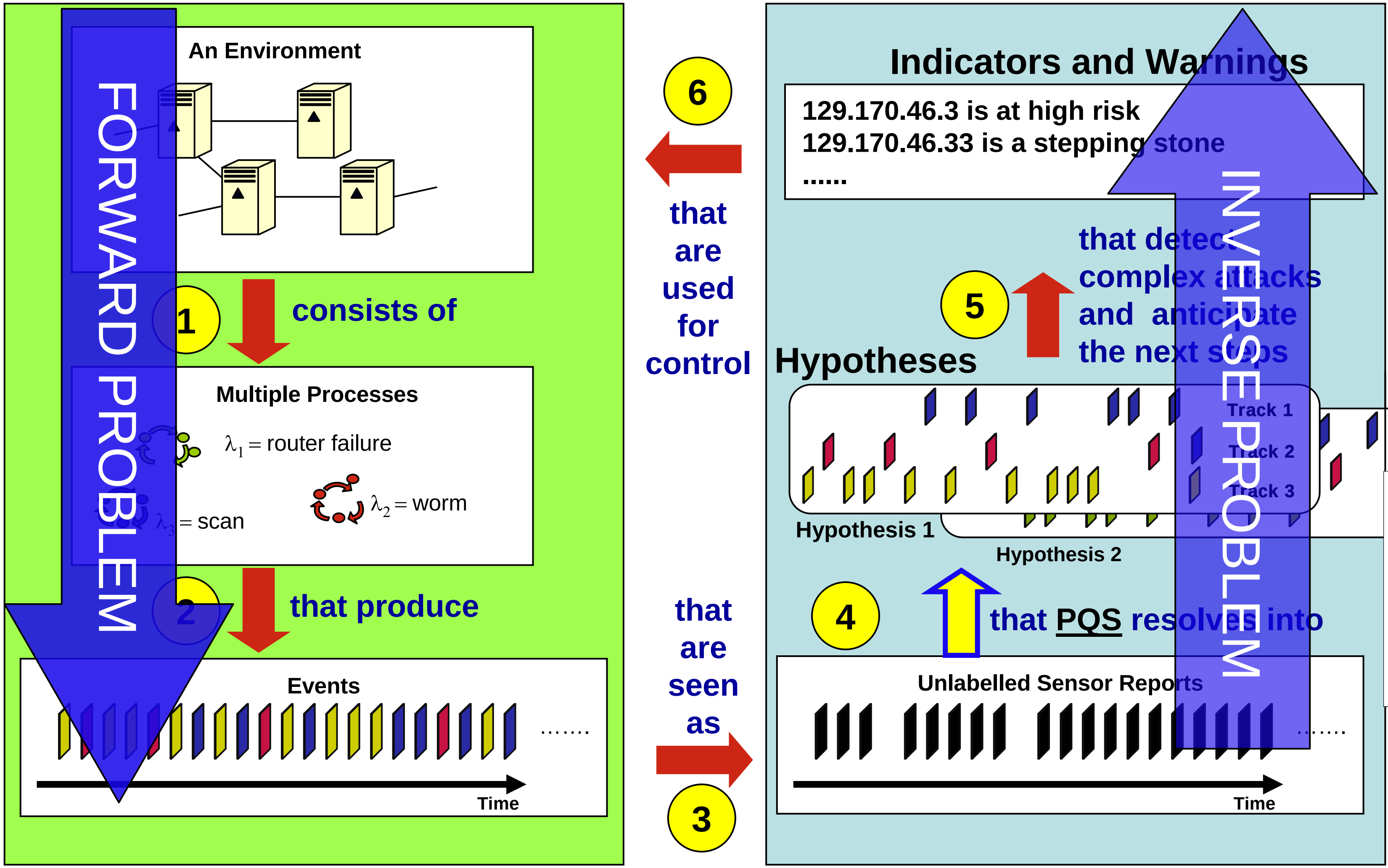
# Process Query System



Implemented for:

Vehicle Tracking
Computer Security
Social Network
Plume Tracking

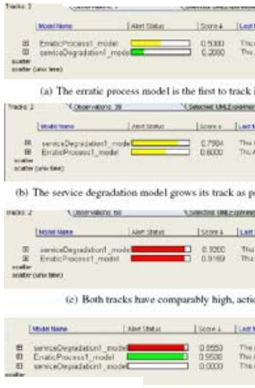
# Cyber Situational Awareness



Real World

Process Detection (PQS)

Sample Console


Tracks: 2

| | Model Name | Alert Status | Score | Last |
|---|---|---|---|---|
| ⊞ | ErraticProcess1_model | | 0.5000 | Thu |
| ⊞ | serviceDegradation1_model | | 0.2000 | Thu |

(a) The erratic process model is the first to track i

Tracks: 2 Observations: 39 Selected UML

| | Model Name | Alert Status | Score | Last |
|---|---|---|---|---|
| ⊞ | serviceDegradation1_model | | 0.7984 | Thu |
| ⊞ | ErraticProcess1_model | | 0.6000 | Thu |

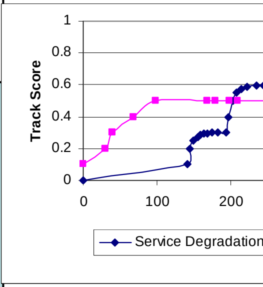
(b) The service degradation model grows its track as p

Tracks: 2 Observations: 50

| | Model Name | Alert Status | Score | Last |
|---|---|---|---|---|
| ⊞ | serviceDegradation1_model | | 0.3000 | Thu |
| ⊞ | ErraticProcess1_model | | 0.9189 | Thu |

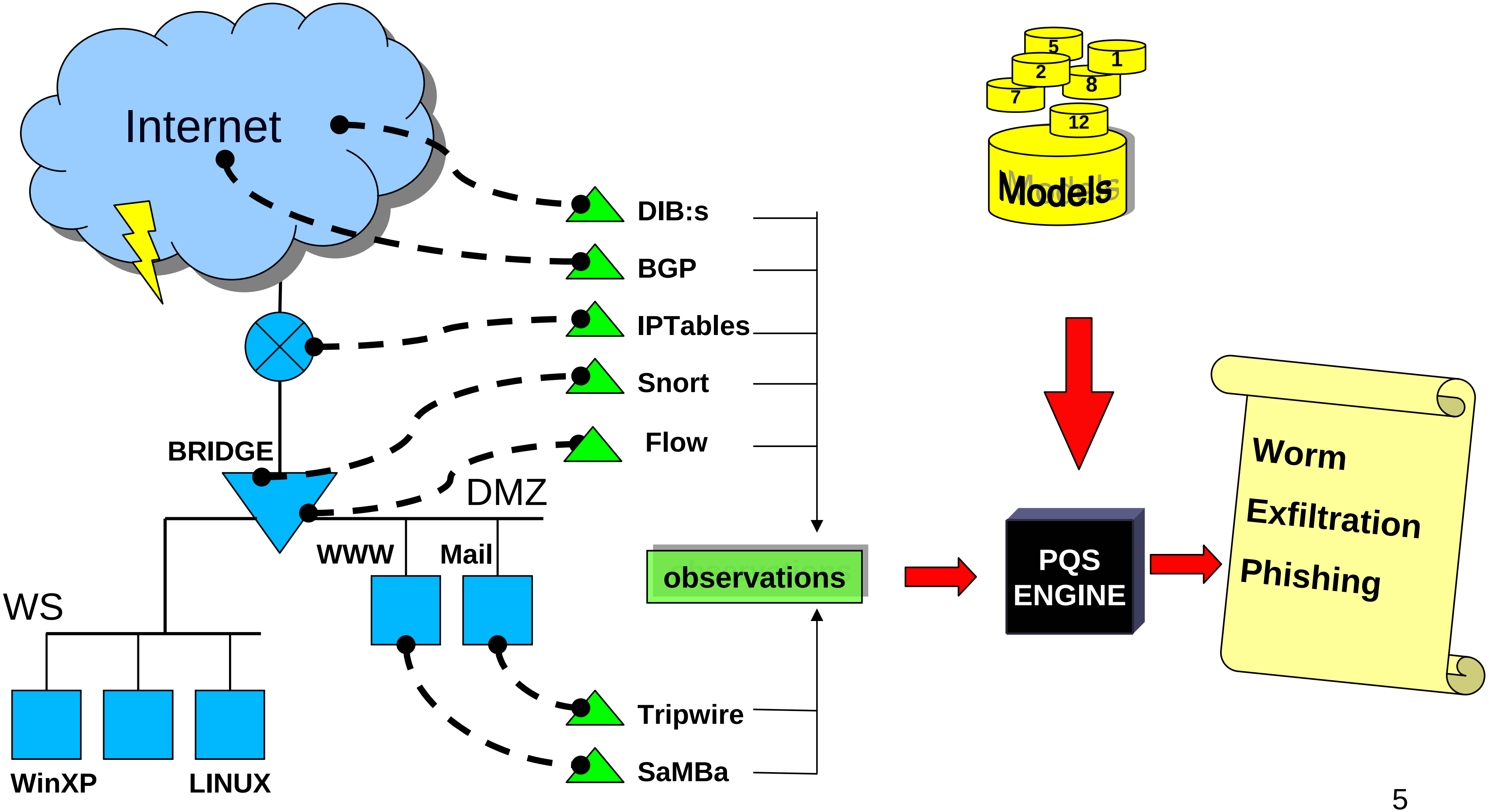
(c) Both tracks have comparably high, actio

| | Model Name | Alert Status | Score | Last |
|---|---|---|---|---|
| ⊞ | serviceDegradation1_model | | 0.5550 | Thu |
| ⊞ | ErraticProcess1_model | | 0.9500 | Thu |
| ⊞ | serviceDegradation1_model | | 0.0000 | Thu |



Track Scores

# PQS in Computer Security



# Sensors and Models

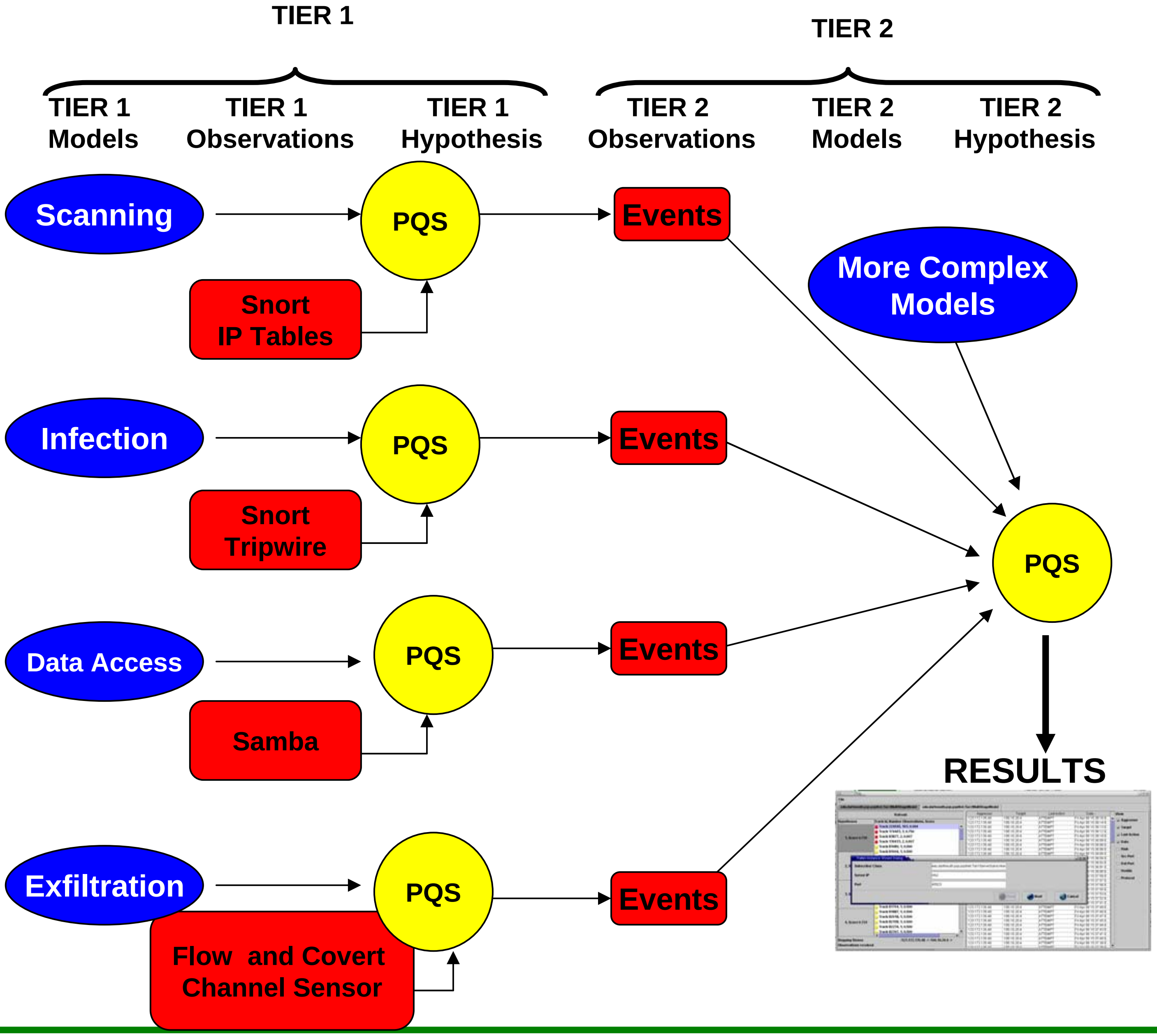
Sensors

| | | |
|---|---|---|
| 1 | DIB:s | Dartmouth ICMP-T3 Bcc: System |
| 2 | Snort, Dragon | Signature Matching IDS |
| 3 | IPtables | Linux Netfilter firewall, log based |
| 4 | Samba | SMB server - file access reporting |
| 5 | Flow sensor | Network analysis ← |
| 6 | ClamAV | Virus scanner |
| 7 | Tripwire | Host filesystem integrity checker |

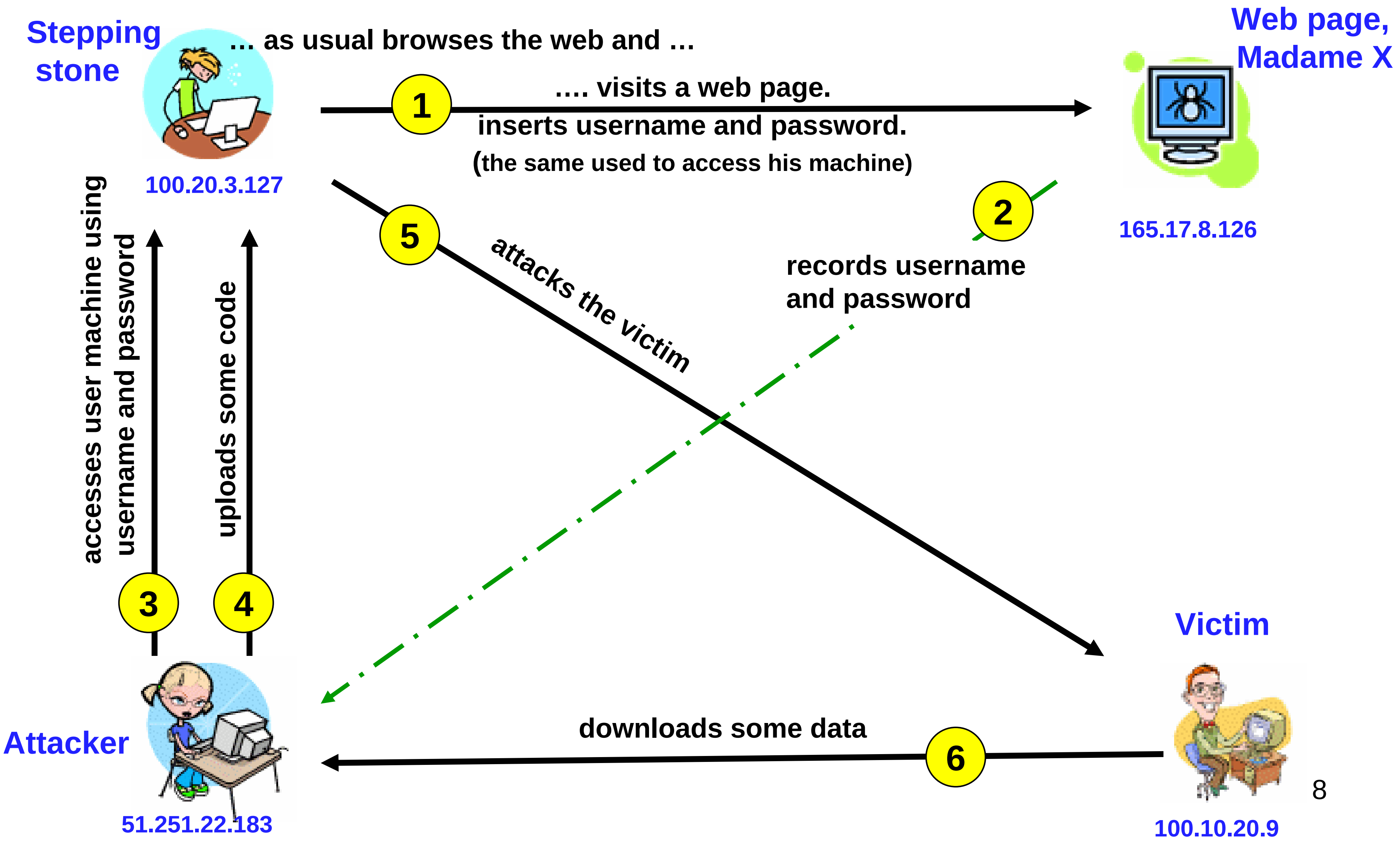
Models

| | |
|---|---|
| 1 | Noisy Internet Worm Propagation – fast scanning |
| 2 | Email Virus Propagation – hosts aggressively send emails |
| 3 | Low&Slow Stealthy Scans – of our entire network |
| 4 | Unauthorized Insider Document Access – insider information theft |
| 5 | Multistage Attack – several penetrations, inside our network |
| 6 | DATA movement |
| 7 | TIER 2 models |

# Hierarchical Architecture



# Multi Stage Attack Example: Phishing



Stepping stone

… as usual browses the web and …

100.20.3.127

1 …. visits a web page.

inserts username and password.

(the same used to access his machine)

Web page, Madame X

165.17.8.126

2 records username and password

5 attacks the victim

3 accesses user machine using username and password

4 uploads some code

Attacker

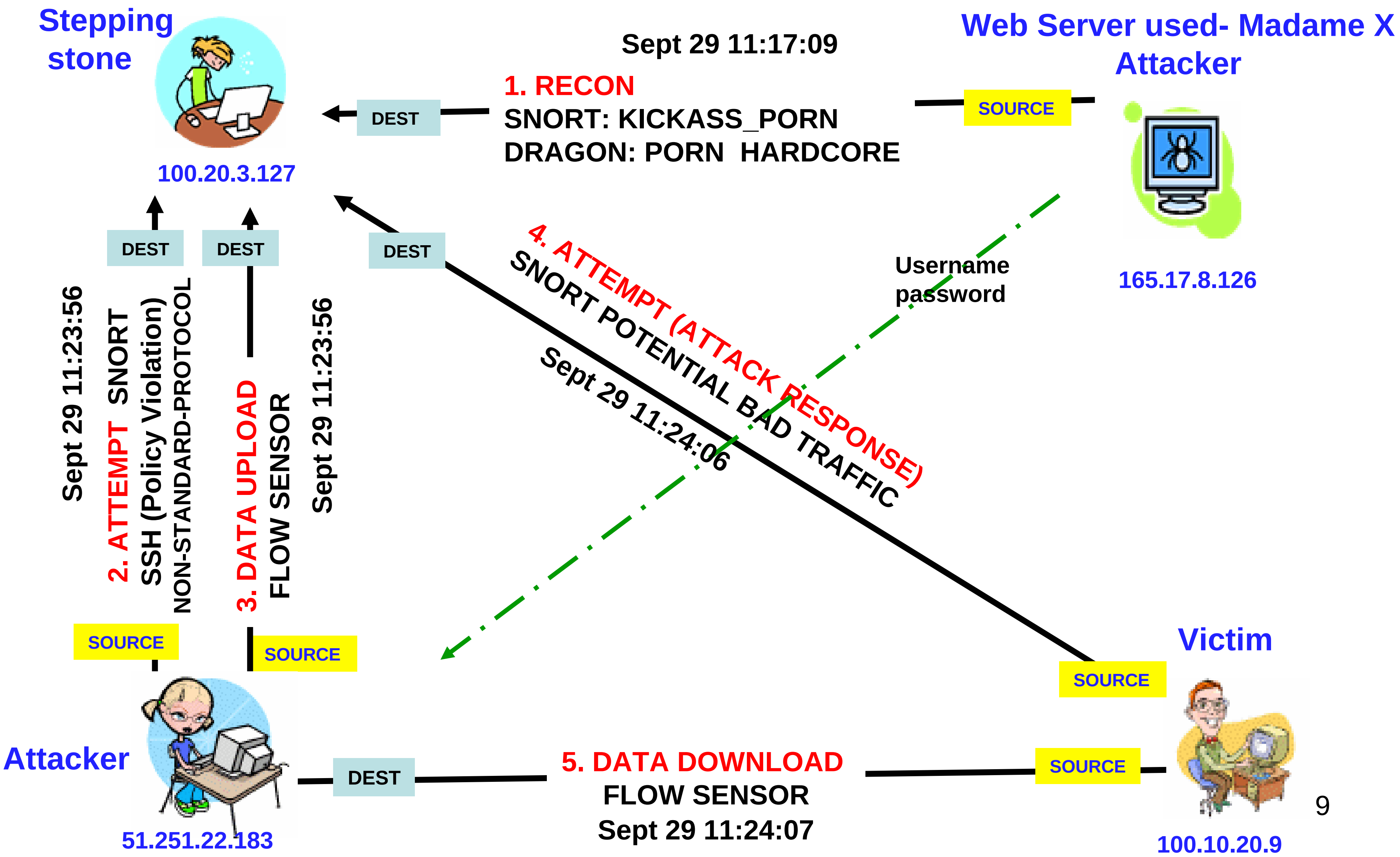
51.251.22.183

6 downloads some data

Victim

100.10.20.9

# Phishing Attack Observables



Stepping stone

100.20.3.127

Web Server used- Madame X Attacker

165.17.8.126

Sept 29 11:17:09

1. RECON
SNORT: KICKASS_PORN
DRAGON: PORN HARDCORE

Username password

Sept 29 11:23:56

2. ATTEMPT SNORT
SSH (Policy Violation)
NON-STANDARD-PROTOCOL

3. DATA UPLOAD
FLOW SENSOR

Sept 29 11:23:56

4. ATTEMPT (ATTACK RESPONSE)
SNORT POTENTIAL BAD TRAFFIC
Sept 29 11:24:06

Victim

100.10.20.9

Attacker

51.251.22.183

5. DATA DOWNLOAD
FLOW SENSOR
Sept 29 11:24:07

# Flow Sensor

Based on the *libpcap* interface for packet capturing.

Packets with the same source IP, destination IP, source port, destination port, protocol are aggregated into the same flow.

- Timestamp of the last packet
- # packets from Source to Destination
- # packets from Destination to Source
- # bytes from Source to Destination
- # bytes from Destination to Source
- Array containing delays in microseconds between packets in the flow

# Two Models Based on the Flow Sensor

## Low and Slow UPLOAD

| Volume | Packets | Duration | Balance | Percentage |
|---|---|---|---|---|
| Tiny: 1-128b<br>Small: 128b-1Kb | 4:10-99<br>5: 100-999<br>6: > 1000 | 4: 1000-10000 s<br>5: 10000-100000 s<br>6: > 100000 s | Out | >80 |

## UPLOAD

| Volume | Packets | Duration | Balance | Percentage |
|---|---|---|---|---|
| Tiny: 1-128b<br>Small: 128b-1Kb<br>Medium: 1Kb-100Kb<br>Large: > 100Kb | 1: one packet<br>2: two pckts<br>3: 3-9<br>4: 10-99<br>5: 100-999<br>6: > 1000 | 0: < 1 s<br>1: 1-10 s<br>2: 10-100 s<br>3: 100-1000 s<br>4: 1000-10000 s<br>5: 10000-100000 s<br>6: > 100000 s | Out | >80 |

# Aggregation

## Flow aggregation.

Recognizing that different flows, apparently totally unrelated, nevertheless belong to the same broader event (activity).

Flows are aggregated from captured network packets.

We aggregate flows into **activities**.

**Example:**

User requests a webpage (all DNS and HTTP flows aggregated)

## Activity aggregation.

Recognizing that similar activities occur regularly at the same time, or dissimilar activities occur regularly in the same sequence.

We correlate activities into **activity groups, patterns**.

**Examples:**

- Nightly backups to all servers (each backup is an activity)
- User requests a sequence of web-pages every morning.

*Packet* = Aggregated Bytes
*Flow* = Correlated Packets
*Activity* = Correlated Flows
*Pattern* = Correlated Activities

# Web Surfing in Detail

1. The **browser communicates with a name server** to translate the server name "www.dartmouth.edu" into an IP Address, which it uses to connect to the server machine.

A FLOW IS INITIATED

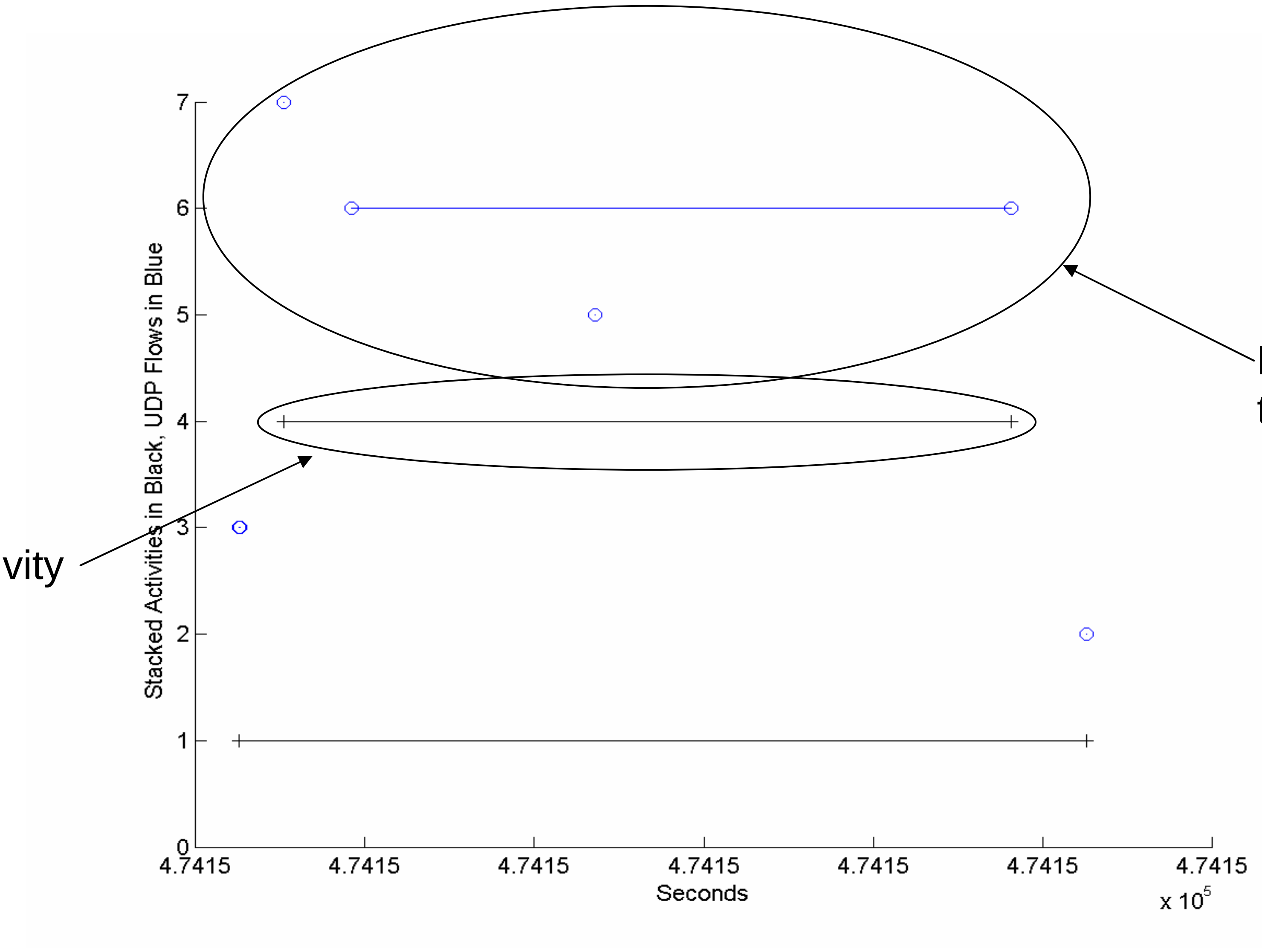
2. The **browser forms a connection to the web server** at that IP address on port 80.

A FLOW IS INITIATED

3. Following the HTTP protocol, the browser sends a GET request to the server, asking for the file "http://www.dartmouth.edu/index.html."
4. The web server sends the HTML text for the Web page to the browser.
5. The browser reads the HTML tags and formatted the page onto your screen.
6. Browser possibly initiates more **DNS requests for media** such as images and video.
7. Browser initiates more **HTTP and/or FTP requests for media.**

MULTIPLE FLOWS ARE INITIATED…

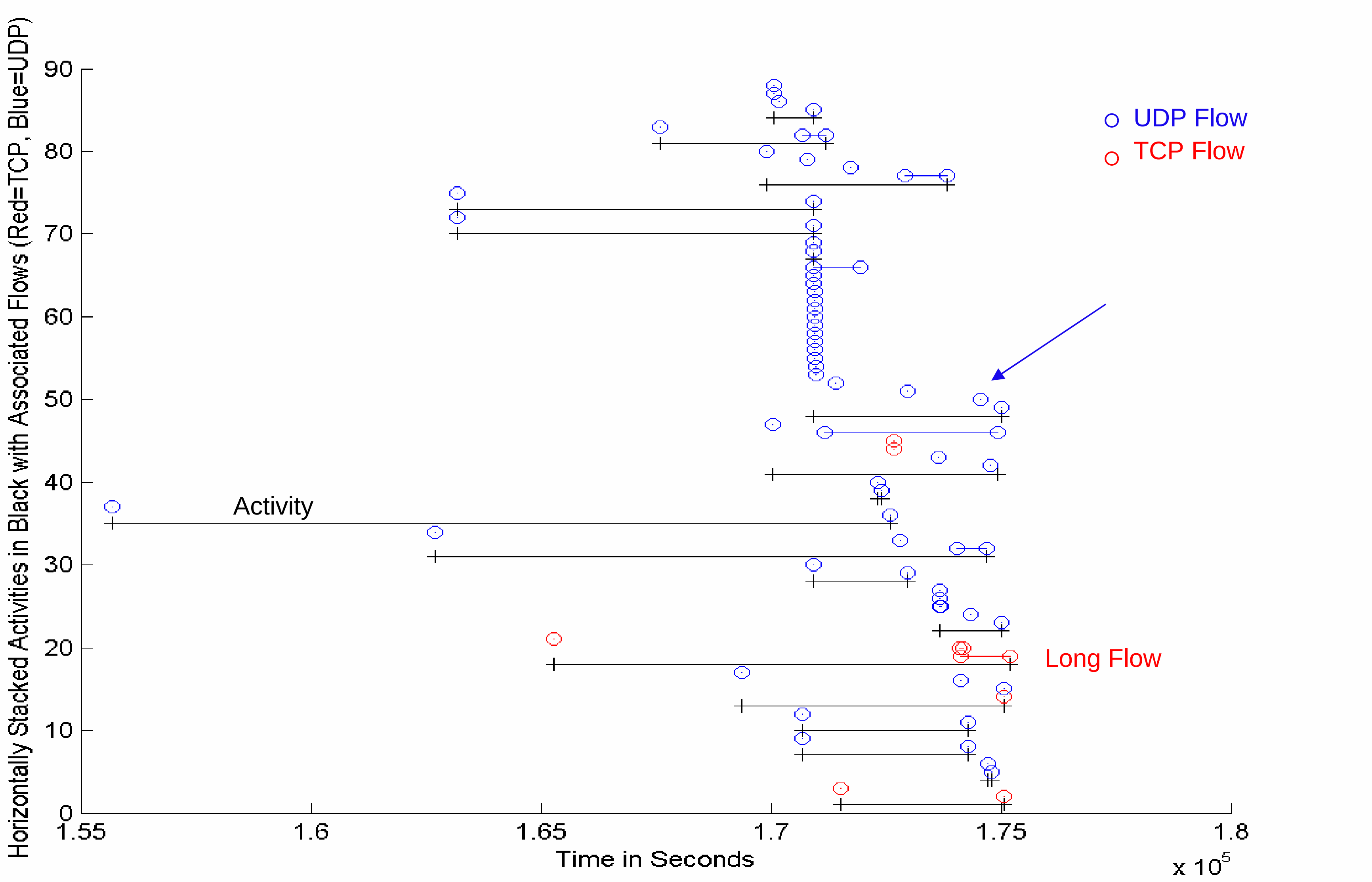
# Resulting Flows and Activity



Flows in the activity

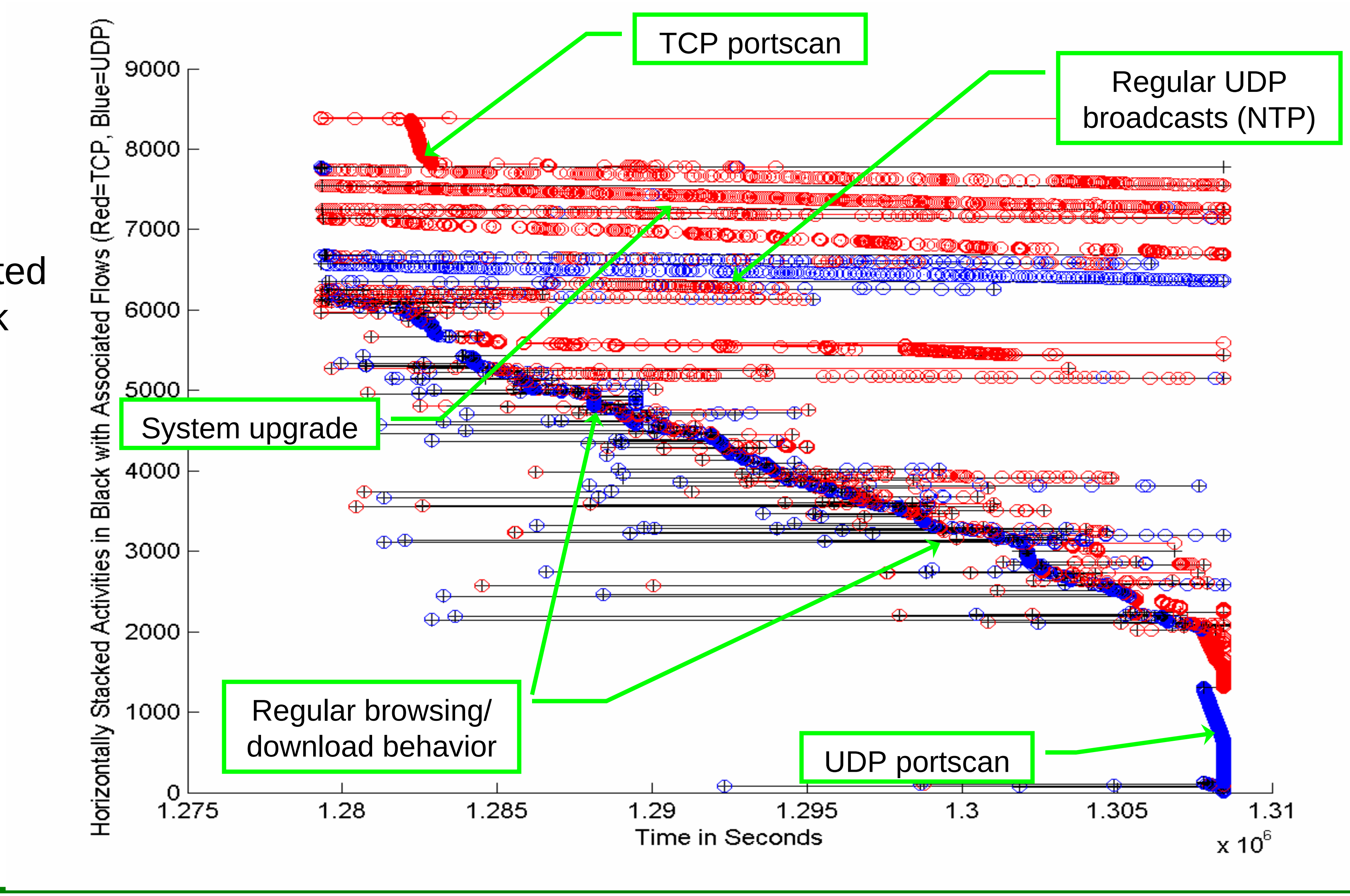
Activity

# Activities and Flows



# Complex Activities ....

Correlated Network Flows Within a LAN

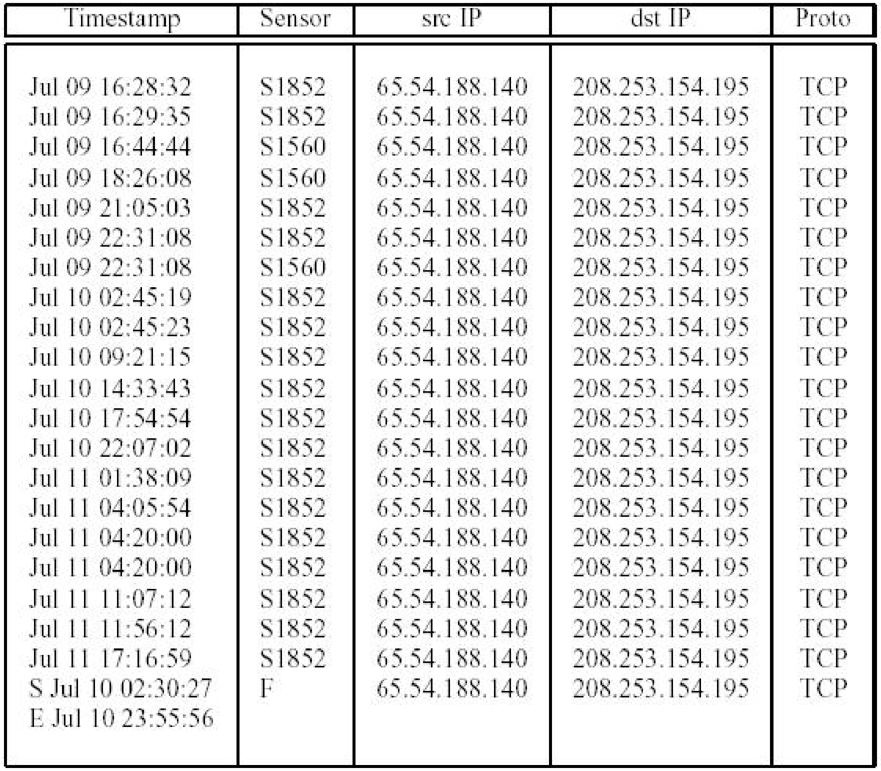


# Packets in a flow triggered IDS alerts

PQS instantiates models based on observation coming from flow and snort sensor.

Snort rule **1560** generates an alert when an attempt is made to exploit a known vulnerability in a web server or a web application.

Snort rule **1852** generates an alert when an attempt is made to access the 'robots.txt' file directly.



| Timestamp | Sensor | src IP | dst IP | Proto |
|---|---|---|---|---|
| Jul 09 16:28:32 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 16:29:35 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 16:44:44 | S1560 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 18:26:08 | S1560 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 21:05:03 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 22:31:08 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 09 22:31:08 | S1560 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 02:45:19 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 02:45:23 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 09:21:15 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 14:33:43 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 17:54:54 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 10 22:07:02 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 01:38:09 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 04:05:54 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 04:20:00 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 04:20:00 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 11:07:12 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 11:56:12 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| Jul 11 17:16:59 | S1852 | 65.54.188.140 | 208.253.154.195 | TCP |
| S Jul 10 02:30:27<br>E Jul 10 23:55:56 | F | 65.54.188.140 | 208.253.154.195 | TCP |

SNORT ALERTS

FLOW

Table 2: A sample track of correlated IDS and Flow events

The flow can be characterized as malicious and further investigation must be done.

# Future Direction

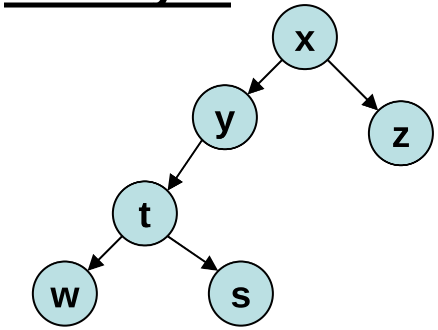
Theoretical approach for clustering aggregated flows.

*Flow* = As defined
*Activity* = Aggregated flows
*Pattern* = Correlated Activities

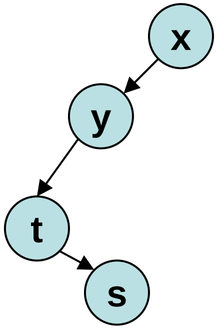
Approach: Graph theory (flows are the nodes and the edges are between correlated nodes).

We are thinking about defining a metric that captures the closeness between two different activities to allow grouping into patterns.

**Activity 1.**



**Activity 2.**

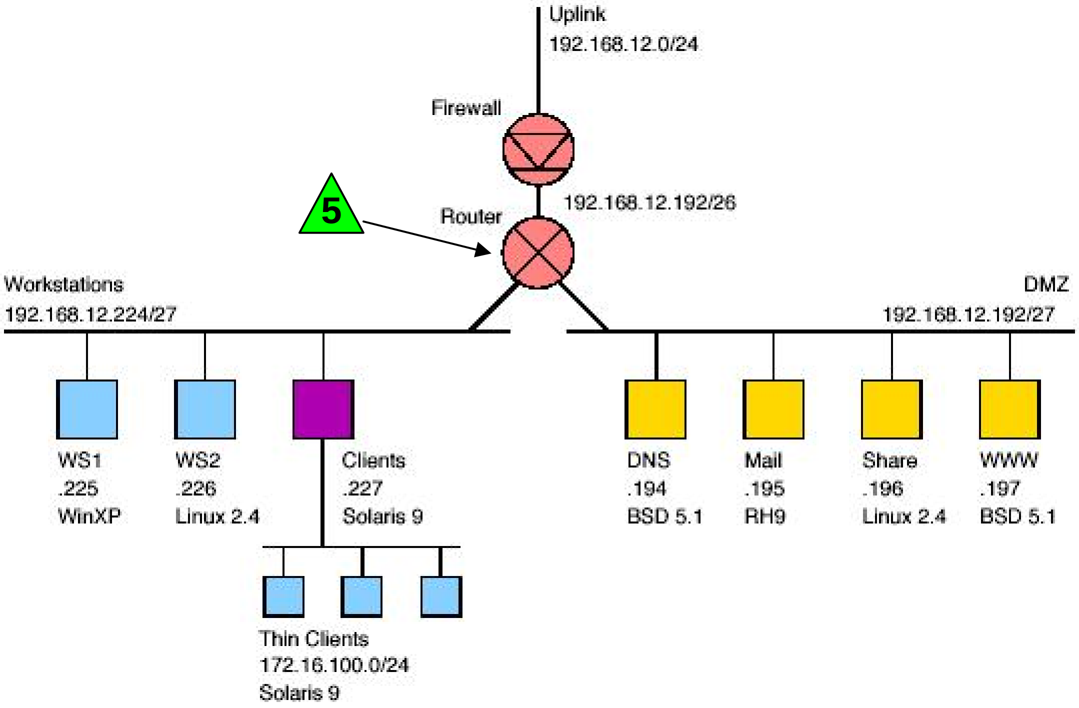


Can they be grouped in one pattern?
Notion of distance between activities.

www.pqsnet.net

agiani@ists.dartmouth.edu

# PQS-Net Network

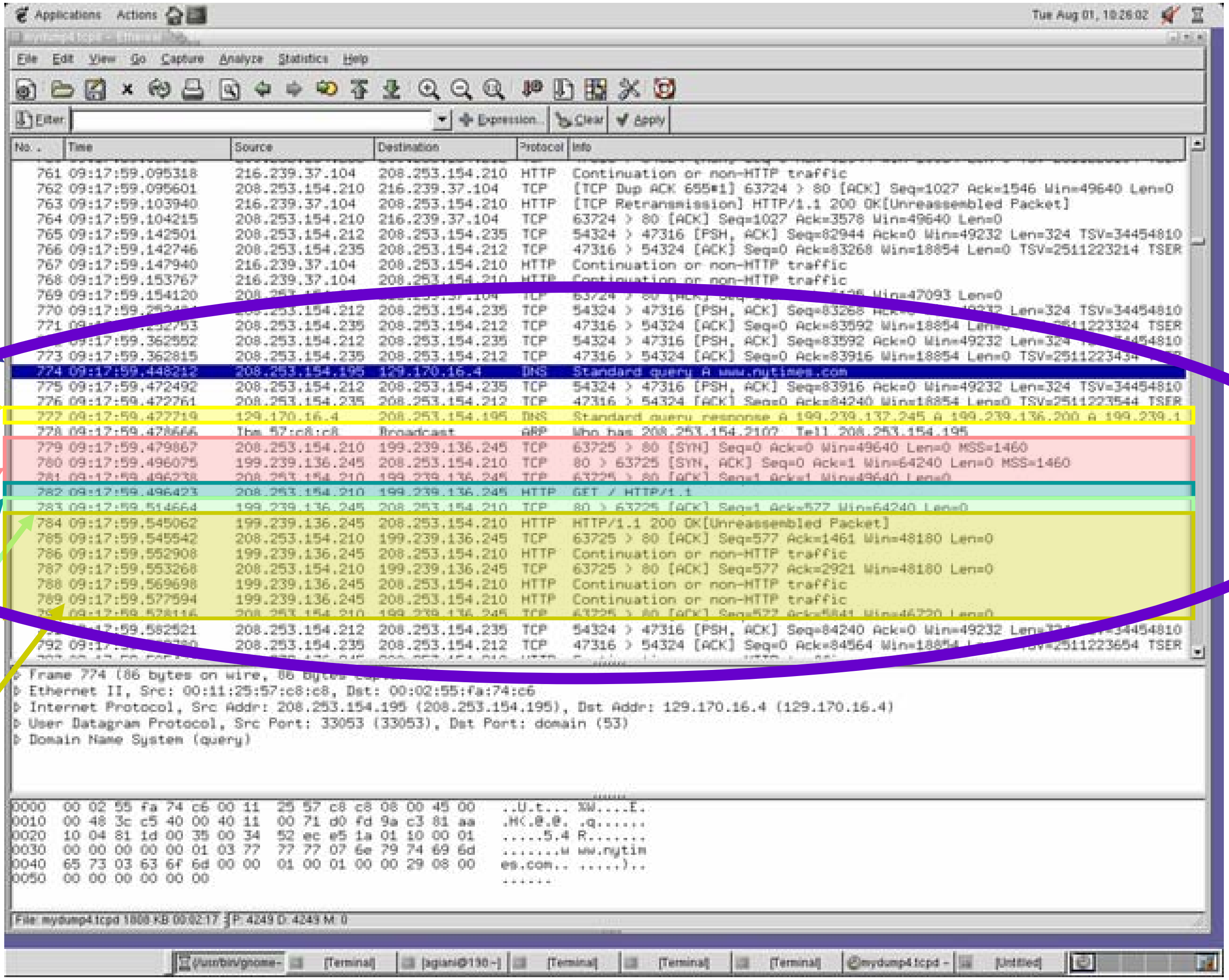


Student and researcher use this network to browse the web, print documents, send upload and download files…

# Web Surfing

| | |
|---|---|
| 208.253.154.210 | host name |
| 208.253.154.195 | dns.pqsnet.net |
| 129.170.16.4 | ns.dartmouth.edu |

1. ns.pqsnet.net requests www.nytimes.com ip address to ns.dartmouth.edu
2. ns.dartmouth.edu returns the ip address – 199.239.136.245
3. TCP three-way handshake between the host machine and the web server.
4. HTTP GET request to 199.239.136.245
5. TCP ACK from the web server
6. Other packets exchanges between the web server and the host



```
No. . Time            Source           Destination      Protocol Info
  761 09:17:59.095318 216.239.37.104   208.253.154.210  HTTP   Continuation or non-HTTP traffic
  762 09:17:59.095601 208.253.154.210  216.239.37.104   TCP    [TCP Dup ACK 655#1] 63724 > 80 [ACK] Seq=1027 Ack=1546 Win=49640 Len=0
  763 09:17:59.103940 216.239.37.104   208.253.154.210  HTTP   [TCP Retransmission] HTTP/1.1 200 OK[Unreassembled Packet]
  764 09:17:59.104215 208.253.154.210  216.239.37.104   TCP    63724 > 80 [ACK] Seq=1027 Ack=3578 Win=49640 Len=0
  765 09:17:59.142501 208.253.154.212  208.253.154.235  TCP    54324 > 47316 [PSH, ACK] Seq=82944 Ack=0 Win=49232 Len=324 TSV=34454810
  766 09:17:59.142746 208.253.154.235  208.253.154.212  TCP    47316 > 54324 [ACK] Seq=0 Ack=83268 Win=18854 Len=0 TSV=2511223214 TSER
  767 09:17:59.147940 216.239.37.104   208.253.154.210  HTTP   Continuation or non-HTTP traffic
  768 09:17:59.153767 216.239.37.104   208.253.154.210  HTTP   Continuation or non-HTTP traffic
  769 09:17:59.154120 208.253.154.210  216.239.37.104   TCP    63724 > 80 [ACK] Seq=        4128 Win=47093 Len=0
  770 09:17:59.25     208.253.154.212  208.253.154.235  TCP    54324 > 47316 [PSH, ACK] Seq=83268 Ack       49232 Len=324 TSV=34454810
  771 09:   .232753     208.253.154.235  208.253.154.212  TCP    47316 > 54324 [ACK] Seq=0 Ack=83592 Win=18854 Len        2511223324 TSER
      09:17:59.362552 208.253.154.212  208.253.154.235  TCP    54324 > 47316 [PSH, ACK] Seq=83592 Ack=0 Win=49232 Len=324     454810
  773 09:17:59.362815 208.253.154.235  208.253.154.212  TCP    47316 > 54324 [ACK] Seq=0 Ack=83916 Win=18854 Len=0 TSV=2511223434
  774 09:17:59.448212 208.253.154.195  129.170.16.4     DNS    Standard query A www.nytimes.com
  775 09:17:59.472492 208.253.154.212  208.253.154.235  TCP    54324 > 47316 [PSH, ACK] Seq=83916 Ack=0 Win=49232 Len=324 TSV=34454810
  776 09:17:59.472761 208.253.154.235  208.253.154.212  TCP    47316 > 54324 [ACK] Seq=0 Ack=84240 Win=18854 Len=0 TSV=2511223544 TSER
  777 09:17:59.477719 129.170.16.4     208.253.154.195  DNS    Standard query response A 199.239.137.245 A 199.239.136.200 A 199.239.1
  778 09:17:59.478666 Ibm_57:c8:c8      Broadcast        ARP    Who has 208.253.154.210?  Tell 208.253.154.195
  779 09:17:59.479867 208.253.154.210  199.239.136.245  TCP    63725 > 80 [SYN] Seq=0 Ack=0 Win=49640 Len=0 MSS=1460
  780 09:17:59.496075 199.239.136.245  208.253.154.210  TCP    80 > 63725 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  781 09:17:59.496238 208.253.154.210  199.239.136.245  TCP    63725 > 80 [ACK] Seq=1 Ack=1 Win=49640 Len=0
  782 09:17:59.496423 208.253.154.210  199.239.136.245  HTTP   GET / HTTP/1.1
  783 09:17:59.514664 199.239.136.245  208.253.154.210  TCP    80 > 63725 [ACK] Seq=1 Ack=577 Win=64240 Len=0
  784 09:17:59.545062 199.239.136.245  208.253.154.210  HTTP   HTTP/1.1 200 OK[Unreassembled Packet]
  785 09:17:59.545542 208.253.154.210  199.239.136.245  TCP    63725 > 80 [ACK] Seq=577 Ack=1461 Win=48180 Len=0
  786 09:17:59.552908 199.239.136.245  208.253.154.210  HTTP   Continuation or non-HTTP traffic
  787 09:17:59.553268 208.253.154.210  199.239.136.245  TCP    63725 > 80 [ACK] Seq=577 Ack=2921 Win=48180 Len=0
  788 09:17:59.569698 199.239.136.245  208.253.154.210  HTTP   Continuation or non-HTTP traffic
  789 09:17:59.577594 199.239.136.245  208.253.154.210  HTTP   Continuation or non-HTTP traffic
  79  09:17:59.578116 208.253.154.210  199.239.136.245  TCP    63725 > 80 [ACK] Seq=577 Ack=5841 Win=46720 Len=0
        :17:59.582521 208.253.154.212  208.253.154.235  TCP    54324 > 47316 [PSH, ACK] Seq=84240 Ack=0 Win=49232 Len=3     =34454810
  792 09:17:                208.253.154.235  208.253.154.212  TCP    47316 > 54324 [ACK] Seq=0 Ack=84564 Win=1885        =2511223654 TSER

▷ Frame 774 (86 bytes on wire, 86 bytes cap
▷ Ethernet II, Src: 00:11:25:57:c8:c8, Dst: 00:02:55:fa:74:c6
▷ Internet Protocol, Src Addr: 208.253.154.195 (208.253.154.195), Dst Addr: 129.170.16.4 (129.170.16.4)
▷ User Datagram Protocol, Src Port: 33053 (33053), Dst Port: domain (53)
▷ Domain Name System (query)

0000  00 02 55 fa 74 c6 00 11  25 57 c8 c8 08 00 45 00   ..U.t... %W....E.
0010  00 48 3c c5 40 00 40 11  00 71 d0 fd 9a c3 81 aa   .H<.@.@. .q......
0020  10 04 81 1d 00 35 00 34  52 ec e5 1a 01 10 00 01   .....5.4 R.......
0030  00 00 00 00 00 01 03 77  77 77 07 6e 79 74 69 6d   .......w ww.nytim
0040  65 73 03 63 6f 6d 00 00  01 00 01 00 00 29 08 00   es.com.. .....)..
0050  00 00 00 00 00 00                                  ......

File: mydump4.tcpd 1808 KB 00:02:17  P: 4249 D: 4249 M: 0
```

All these network connections are related to the same host activity.

# The Effect of Packet Sampling on Anomaly Detection

Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Anukool Lakhina, Martin May

***Abstract*—**

**Packet sampling methods such as Cisco's NetFlow are widely employed by large networks to reduce the amount of traffic data measured. A key problem with packet sampling is that it is inherently a lossy process, discarding (potentially useful) information. In this paper, we empirically evaluate the impact of sampling on anomaly detection. Starting with *unsampled* traffic records collected during the Blaster worm outbreak, we reconstruct the underlying packet trace and simulate packet sampling at increasing rates. We then use our knowledge of the Blaster anomaly to build a baseline of normal traffic (without Blaster), against which we can measure the anomaly size at various sampling rates. This approach allows us to evaluate the impact of packet sampling on anomaly detection without being restricted to (or biased by) a particular anomaly detection method.**

**We find that packet sampling does not disturb the anomaly size when measured in volume metrics such as the number of bytes and number of packets, but grossly biases the number of flows. However, we find that recently proposed entropy-based summarizations of packet and flow counts are affected less by sampling, and expose the Blaster worm outbreak even at higher sampling rates. Our findings suggest that entropy summarizations are more resilient to sampling than volume metrics. Thus, while not perfect, sampling still preserves sufficient distributional structure, which when harnessed by tools like entropy, can expose hard-to-detect scanning anomalies.**

## I. Introduction

Traffic sampling has emerged as the dominant means to summarize the vast amount of traffic data continuously collected for network monitoring. The most prevalent and widely-deployed method of sampling traffic is *packet sampling*, where a router inspects every $n$-th packet (uniformly at random), and records its features (addresses, ports, protocol, and flags).

But, while being attractive because of efficiency and availability, sampling is inherently a lossy process, where many packets are discarded without inspection. Thus sampled traffic is an incomplete and more importantly, a biased approximation of the underlying traffic trace, as small flows are likely to be missed entirely. Previous work has largely focused on analyzing this bias, devising better sampling strategies [3], and recovering statistics (moments and distribution) of the underlying traffic trace using inference [5, 6, 8].

There is comparatively little previous work on how sampling impacts network monitoring applications, such as anomaly detection. Indeed sampled traffic views have recently been used for signature-based security analysis and anomaly detection with considerable success [10, 12]. But, little is known about the fidelity of the sampled stream for these applications, and basic questions remain unanswered; for example: how complete are the detections revealed by these methods on sampled traffic? and: what kind of anomalies are discarded by packet sampling? Clearly signature-based security detection schemes – which look for specific, often detailed, packet-level connection patterns in traffic – will be very sensitive to missing packets, and will be impacted by packet sampling. However, the impact of packet sampling on anomaly detection is less clear. This is because anomaly detection is concerned not with finding specific packet patterns in sampled traffic, but rather with exposing incidents that deviate significantly from typical traffic behavior. Indeed, anomaly detection methods work by building models for "normal" traffic over a period of time (typically days to week), and then reporting events that are outliers (according to some distance measure) from this baseline normal model. Therefore, for packet sampling to impact anomaly detection it has to either: (1) disturb the baseline model of normal traffic drastically, or (2) dwarf the anomaly significantly so that it does not deviate from the baseline in any detectable manner.

In this paper, we empirically study the impact of packet sampling on anomaly detection, focusing on how sampling dwarfs a known anomaly, when compared to a baseline. For our evaluation, we rely on a unique week-long dataset of *unsampled* traffic records with the Blaster worm anomaly, collected from backbone routers of a national ISP. We then simulate packet sampling to construct sampled views of the same traffic trace and ask how the sampled view differs from the original trace, *from an anomaly detection standpoint*. Rather than focus on a particular anomaly detection method, we adopt a general methodology. Because we know the exact characteristics of the anomaly in our trace, we can build the ideal normal baseline, that all anomaly detection methods would strive to build. We then study the size of the worm anomaly, which is measured as the distance from this ideal baseline, at increasing sampling rates.

As a starting point, we investigate how packet sampling impacts the three principal volume metrics (number of bytes, packets and flows), which have been used widely by many detection methods [1, 2, 11]. We find that packet sampling impacts byte counts and packet counts little, but impacts flow counts radically. This finding suggests that anomalies that impact packet and byte volume only will stand out even in sampled traffic, but anomalies that impact flow counts alone (such as the Blaster worm in our data) are likely to be discarded by packet sampling. Therefore detection schemes based on flow volume alone are likely to be inadequate for sampled traffic.

In addition to volume metrics, we also study the impact of

D. Brauckhoff, B. Tellenbach, A. Wagner and M. May are with the Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland; email: {brauckhoff, tellenbach, wagner, may}@tik.ee.ethz.ch. A. Lakhina is with the Department of Computer Science at Boston University; email: anukool@cs.bu.edu

packet sampling on *feature entropy* metrics [12, 14]. The authors of [12] showed that changes in distributions of traffic features (ports and addresses), when summarized by entropy, reveal a broad spectrum of anomalies. We evaluated how effective entropy is at exposing anomalies at increasing sampling rates. Our results here are surprising: we find that while flow volume is grossly impacted by packet sampling, flow entropy is disturbed little. In particular, the Blaster worm in our data when measured in flow counts is dwarfed significantly and virtually undetectable at higher sampling rates, but the worm remains largely unaffected by sampling when measured from a baseline entropy. Thus, the structure of the Blaster worm, as captured by entropy, is preserved even at high sampling rates of 1 in 1000. Our findings provide hope that even though packet sampling produces imperfect traffic views for anomaly detection, there are metrics (such as entropy) that allow us to harness useful information in sampled traces.

The rest of this paper is organized as follows. We next provide an overview of our methodology. In Section 3, we introduce our anomaly detection model and study the impact of packet sampling on detecting flow-based anomalies. In Section 4, we conclude and outline directions for future work.

## II. Methodology

In order to systematically evaluate the impact of *packet* sampling on anomaly detection, one requires *packet*-level traces (at various sampling rates) that ideally meet two criteria: (1) the traces contain known anomalies, and (2) the traces span a long duration (days to week). Known anomalies make evaluation simpler, as ground truth is known a priori. And, longer traces are needed since many anomaly detection methods require a considerable training period in order to profile the normal traffic behavior. Unfortunately, legal requirements (data protection) and technical limitations (storage space), make it difficult to collect such detailed packet-level data.

To circumvent the lack of suitable long-term packet traces, we instead decided to work with unsampled flow records, developed a method to reconstruct packet-level traces from these flow traces.

### A. Reconstructing Packet Traces

Our method to reconstruct the packet traces takes (unsampled) NetFlow records from the Swiss Academic and Research Network (SWITCH) [13] as input and generates the corresponding packet traces. The output format of the packet traces is again flow records with ”flows” that contain only one packet and that have the same start- and end-time. In contrast to real NetFlow records, the packet traces contain ”flows” that are sorted according to their start time.

The reconstruction algorithm processes the flows in the order as they are stored in the flow traces. For each of these flows it does the following: First, the size of the packet is calculated by dividing the total number of bytes $B$ by the number of packets $N$ in the corresponding flow. Afterwards, the time stamp of the packet is randomly selected within flow bounds and with a resolution of one millisecond. With this, the expected size of a packet in the flow is equal to $B/N$ and the expected number of transferred bytes per millisecond is $N/M$.

Furthermore, by choosing the same packet size for all packets, we preserve (on average) the often assumed (e.g., [7], [9]) constant throughput property of flows even if they are split over two intervals. Recently, the authors of [15] presented empirical evidence demonstrating that the constant throughput property is a good approximation of the behavior of large flows (heavy hitter, elephant flows) while still being a reasonalbe approximation for small ones (mice flows).

### B. Effects of Sampling on Byte, Packet, and Flow Metrics

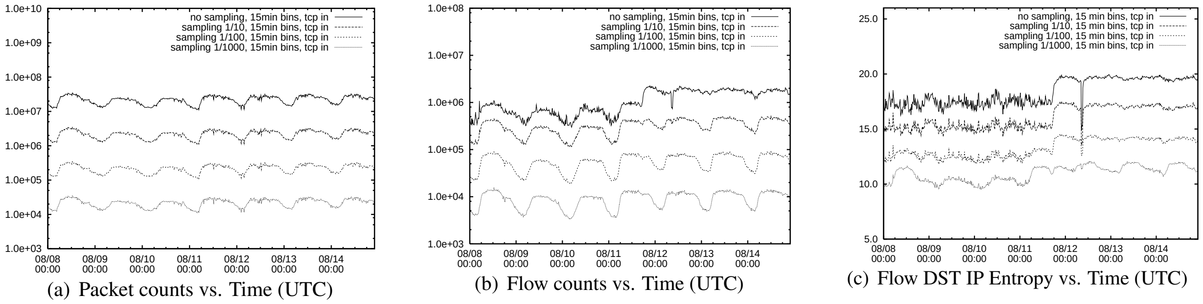
Having reconstructed the packet traces from our NetFlow data, we can now look at how timeseries of volume and feature entropy metrics are impacted by packet sampling. Therefore, we sampled our one-week data set at four different sampling rates of 1 out of 10, 1 out of 100, 1 out of 250, and 1 out of 1000. The sampling method we applied is random probabilistic packet sampling. Thus, sampling at a rate of $q$ we independently select each packet with a probability of $q$ or discard it with a probability of $1 - q$. Subsequently, we computed the timeseries of volume metrics (byte, packet, and flow counts), and feature entropy metrics (packet and flow entropy of IP addresses and port numbers).

To illustrate the following discussion on sampling effects, a selection of meaningful timeseries is depicted in Fig. 1. As expected, Fig. 1(a) shows that packet counts are not disturbed by packet sampling. The unsampled values can simply be estimated by multiplying the sampled value with a factor of $1/q$. This is due to the fact that the variation of packet sizes by a factor of 100 (between 40 and 1500 Bytes) is very small compared to the overall number of Bytes ($\approx 10^{10}$) within one interval of 15 minutes. On the contrary, flow counts are heavily disturbed by packet sampling even at a sampling rate as low as 1 out of 10 (see Figure 1(b)). This can be explained with the fact that small flows (with only few packets) are sampled with a smaller probability compared to larger flows [6].

More interestingly, flow entropy metrics (Fig. 1(c)) are well preserved even at higher sampling rates. Though we see that packet sampling disturbs entropy metrics (the unsampled value cannot easily be computed from the sampled value as for byte and packet counts), the main traffic pattern is still visible in the sampled trace. This insight was the main motivation for this work.

## III. Impact of Sampling on Anomaly Detection

In this section, we study the impact of sampling on anomaly detection methods. Rather than apply a particular anomaly detection method on sampled views of traffic, we adopt a more general strategy based on the observation that fundamentally all anomaly detection methods must first define “normal” behavior; anomalies then become deviations from this baseline behavior. So if we can build the “perfect” baseline (an objective that all anomaly detection methods will strive towards), we can gain insight into the fundamental impact of packet sampling on *any* anomaly detection method, in particular, the detection schemes with the “best” models for baseline behavior.

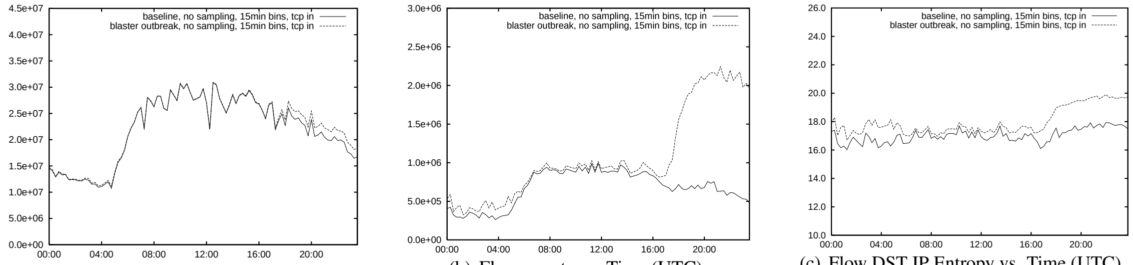


(a) Packet counts vs. Time (UTC)

(b) Flow counts vs. Time (UTC)

(c) Flow DST IP Entropy vs. Time (UTC)

Fig. 1. Impact of Sampling on Timeseries of Selected Metrics.



(a) Packet counts vs. Time (UTC)

(b) Flow counts vs. Time (UTC)

(c) Flow DST IP Entropy vs. Time (UTC)

Fig. 2. Baselines for Selected Metrics.

### *A. Determining the Baseline*

Since our intention is to analyze the effect of sampling on anomaly detection, we need to quantify and measure the factor by which sampling disturbs a certain metric.

An accurate method to determine the level of disturbance is to measure the distance (or normalized distance) between "normal" traffic (from hereon called baseline) and "abnormal" traffic (traffic containing network anomalies). The difficulty, however, as for every anomaly detection method is to correctly determine this baseline. For our study, we have the huge advantage that we know the Blaster anomaly in our trace very well. Thus, we are able to construct an "ideal" baseline by removing the traffic that constitutes to the anomaly. In our case, that is removing all traffic that matches a Blaster heuristic: all packets with destination port 135 and packet sizes of 40, 44, or 48 are removed.

The baseline and the original unsampled trace are depicted for packet counts (Figure 2(a)), flow counts (Figure 2(b)), and flow destination IP address entropy (Figure 2(c)). While packet counts do only show a minor increase in distance before and after the Blaster outbreak, the other three metrics indicate a more drastic and visible change.

### *B. Measuring Anomaly Size*

Having constructed the baselines and packet traces for different sampling rates and metrics, we now answer the question: How is anomaly detection impacted by packet sampling? To address this question, we measure the *anomaly size* at different sampling rates instead of focusing on a particular anomaly detection method. We define anomaly size as the distance between a sampled view $\mathbf{x}$ and the corresponding baseline $\hat{\mathbf{x}}$. We determine the anomaly size, by measuring deviation from the baseline at each timebin using two distance measures:

- the relative difference, defined as: $(\mathbf{x}-\hat{\mathbf{x}})/\hat{\mathbf{x}}$
- the $l_2$ difference: $\sqrt{\sum((\mathbf{x}-\hat{\mathbf{x}})^2)/\sum(\hat{\mathbf{x}}^2)}$

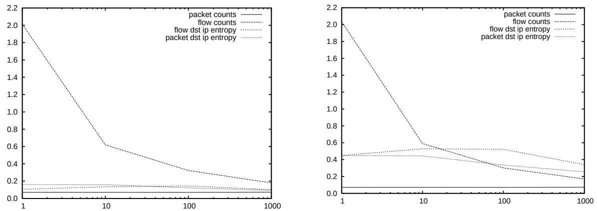
For the relative difference, we computed the distance for each interval individually, and afterwards averaged over a time period starting from the Blaster outbreak on 11/08/2006 at 17:00 until midnight of the same day. For the $l_2$ difference, we computed the sum over the same period for all distances, which was then normed over the sum of all baseline values in this period. Doing this is reasonable since we observe more or less constant Blaster traffic for the whole period (see Fig. 1).

In Fig. 3 we plot the sampling rate vs. the relative difference as well as the sampling rate vs. the $l_2$ difference for packet counts, flow counts, flow destination IP entropy, and packet destination IP entropy. The figure shows four curves, one for each metric under investigation, at each sampling rate. For the flow count metrics the relative difference as well as the $l_2$ difference decrease drastically when sampling is applied.

Packet counts, in contrast, are not impacted by packet sampling and consequently the relative difference as well as the $l_2$ difference for packet counts remain constant. However, the problem with packet counts is that Blaster-type anomalies which usually represent only a very small fraction of all packets (less than 1% in our backbone trace) are not very visible even in the unsampled data traces.

The flow and the packet entropy curves stand in sharp contrast to flow counts. The relative difference as well as the $l_2$ difference decrease only very slightly even for sampling rates as high as 1 out of 1000 for both the entropy metrics, implying that the size of the Blaster worm remains unaffected when viewed using entropy.

To summarize, our results collectively demonstrate that entropy-based metrics have two key benefits over volume-based metrics: (1) they capture the Blaster worm in unsampled traffic, even though the Blaster worm is not clearly visible in packet and byte counts; and more importantly: (2) they are impacted little by sampling when compared to flow counts.



(a) Relative Difference (b) Relative $l_2$ Difference

Fig. 3. Anomaly Size (measured as deviation from the baseline) vs. Sampling Rates for four metrics.

### C. Metric Sensitivity to Anomaly Intensity

To evaluate the sensitivity of entropy towards sampling, we use the given trace and attenuate or amplify the strength of the Blaster anomaly signal. To amplify the Blaster anomaly, we duplicate the attack packets by a factor of 2; for an attenuated attack, we keep only 50%, 20%, and 10% of the attack packets in the p

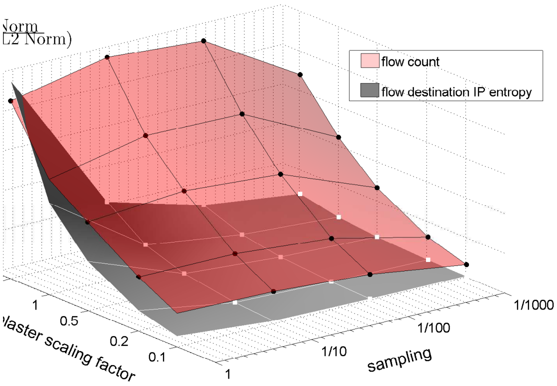


Fig. 4. Normalized anomaly deviation from the baseline for flow counts and flow entropy across increasing sampling rates and different intensities.

Figure 4 presents the anomaly size ($l_2$ difference from the baseline) as captured in two metrics, the flow count (dark gray) and the flow entropy (light gray), across increasing sampling rates and different intensities[1]. It provides considerable insight into the efficacy of flow counts and flow entropy in exposing the Blaster anomaly at various intensities and at various sampling rates.

As expected, the stronger the anomaly the more easily the anomaly will be detected for both metrics. But, flow counts decrease sharply as the Blaster worm is attenuated, even with unsampled traffic. Moreover, this decrease in flow counts is even sharper as the sampling rate increases. In contrast, flow entropy decreases remarkably slow, both with increasing sampling rate and for varying intensities of the Blaster attack.

We conclude from this figure that flow entropy is far more robust to packet sampling than simple flow count based summaries, when exposing the Blaster worm at various intensities.

## IV. Conclusion

In this paper, we empirically evaluated the impact of packet sampling on anomaly detection. With a week-long dataset of unsampled traffic records containing the Blaster worm, we employed a general detection methodology (the deviation from an idealized baseline) to evaluate the fidelity of sampled traffic in exposing anomalies.

Our first finding is somewhat expected: we found that packet sampling produces accurate estimates of byte and packet counts (when compared to the underlying trace). However, packet sampling produces grossly inaccurate estimates of flow counts. Thus, anomalies that only impact packet counts or byte counts, are likely to be visible in sampled views, but anomalies that impact flow counts (such as the Blaster worm in our data) will not be visible.

We then evaluated the effect of packet sampling on *feature entropy*. Surprisingly, we found that while the Blaster worm is entirely undetectable in flow counts of sampled traces, it is visible in flow entropy. While sampled traffic views are necessarily incomplete and imperfect, they are not completely useless; in fact, this paper shows that sampled traffic has utility for anomaly diagnosis, if it is analyzed using the appropriate metrics, such as entropy. The results presented in this paper open up new directions for research on devising detection metrics that are robust to packet sampling.

## References

[1] Barford, P., Kline, J., Plonka, D., and Ron, A. A signal analysis of network traffic anomalies. In *Internet Measurement Workshop* (Marseille, November 2002).

[2] Brutlag, J. Aberrant behavior detection in timeseries for network monitoring. In *USENIX LISA* (New Orleans, December 2000).

[3] Choi, B.-Y., Park, J., and Zhang, Z.-L. Adaptive random sampling for total load estimation. In *IEEE International Conference on Communications* (2003).

[4] Cisco NetFlow. At `www.cisco.com/warp/public/732/Tech/netflow/`.

[5] Duffield, N., Lund, C., and Thorup, M. Properties and prediction of flow statistics from sampled packet streams. In *ACM SIGCOMM Internet Measurment Workshop* (2002).

[6] Duffield, N., Lund, C., and Thorup, M. Estimating Flow Distributions from Sampled Flow Statistics. In *ACM SIGCOMM* (Karlsruhe, August 2003).

[7] Estan, C., and Varghese, G. New directions in traffic measurement and accounting. In *In Proceedings of the 2001 ACM SIGCOMM Internet Measurement Workshop* (San Francisco, CA, 2001), pp. 75–80.

[8] Hohn, N., and Veitch, D. Inverting Sampled Traffic. In *Internet Measurement Conference* (Miami, October 2003).

[9] Jung, J., Krishnamurthy, B., and Rabinovich, M. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *In Proceedings of the International World Wide Web Conference* (2002), pp. 252–262.

[10] Kim, M.-S., Kang, H.-J., Hung, S.-C., Chung, S.-H., and Hong, J. W. A Flow-based Method for Abnormal Network Traffic Detection. In *IEEE/IFIP Network Operations and Management Symposium* (Seoul, April 2004).

[11] Lakhina, A., Crovella, M., and Diot, C. Diagnosing Network-Wide Traffic Anomalies. In *ACM SIGCOMM* (Portland, August 2004).

[12] Lakhina, A., Crovella, M., and Diot, C. Mining Anomalies Using Traffic Feature Distributions. In *ACM SIGCOMM* (Philadelphia, August 2005).

[13] SWITCH. Swiss academic and research network. `http://www.switch.ch/`, 2006.

[14] Wagner, A., and Plattner, B. Entropy based worm and anomaly detection in fast ip networks. In *In Proceedings of the STCA security workshop / WETICE 2005* (2005).

[15] Wallerich, J., Dreger, H., Feldmann, A., Krishnamurthy, B., and Willinger, W. A methodology for studying persistency aspects of internet flows. *SIGCOMM Comput. Commun. Rev. 35*, 2 (2005), 23–36.

[1] For presentation purposes, we normalized each surface by the maximum size for that metric, so that the size of the anomaly for each metric falls between 0 and 1.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Impact of Packet Sampling on Anomaly Detection Metrics

Daniela Brauckhoff*, Bernhard Tellenbach*, Arno Wagner*, **Anukool Lakhina ****, Martin May*

*ETH Zurich, ** Boston University

# Motivation

- The general opinion about sampling:
  - With sampling valuable information lost about anomalies
  - But sampling needs to be used anyway...
    - Cannot get unsampled netflow from some routers

- Interesting questions arise:
  - *How much* information is actually lost?
  - Are all *anomalies* equally affected by sampling?
  - Are all *detection metrics* equally affected by sampling?
  - At which *sampling rate* is a certain anomaly still detectable?
  - Can we estimate the *original anomaly size* from a sampled view?

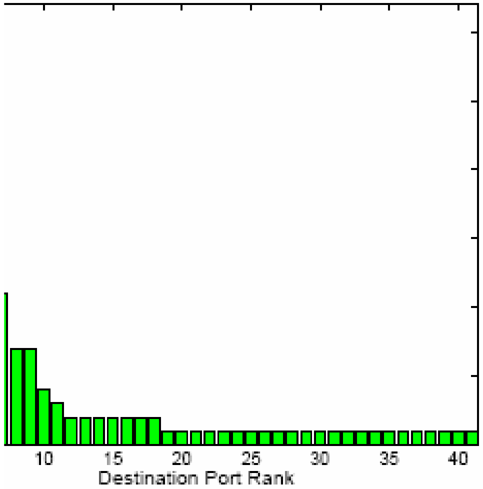
# Data & Experiments

- A week-long dataset of ***unsampled Netflow records*** from a backbone router of a national ISP
- Known Blaster outbreak in our data

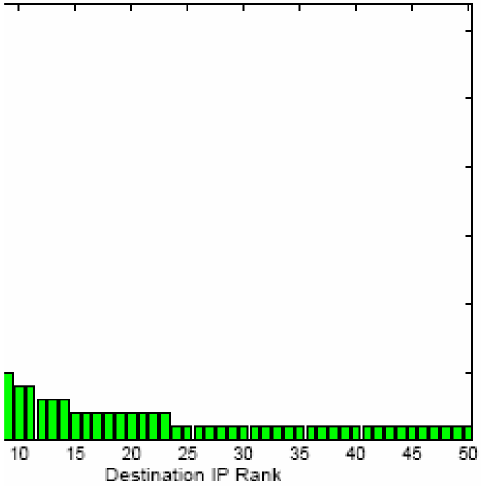
- **Goal:** Study impact of packet sampling on Blaster worm
  - Focus on visibility of Blaster worm
  - Focus on anomaly detection metrics
    - **Bytes, Packets, Flows, Traffic Features**, ...

# Entropy as a Detection Metric [LCD:SIGCOMM05]

**Dispersed Histogram**
High Entropy



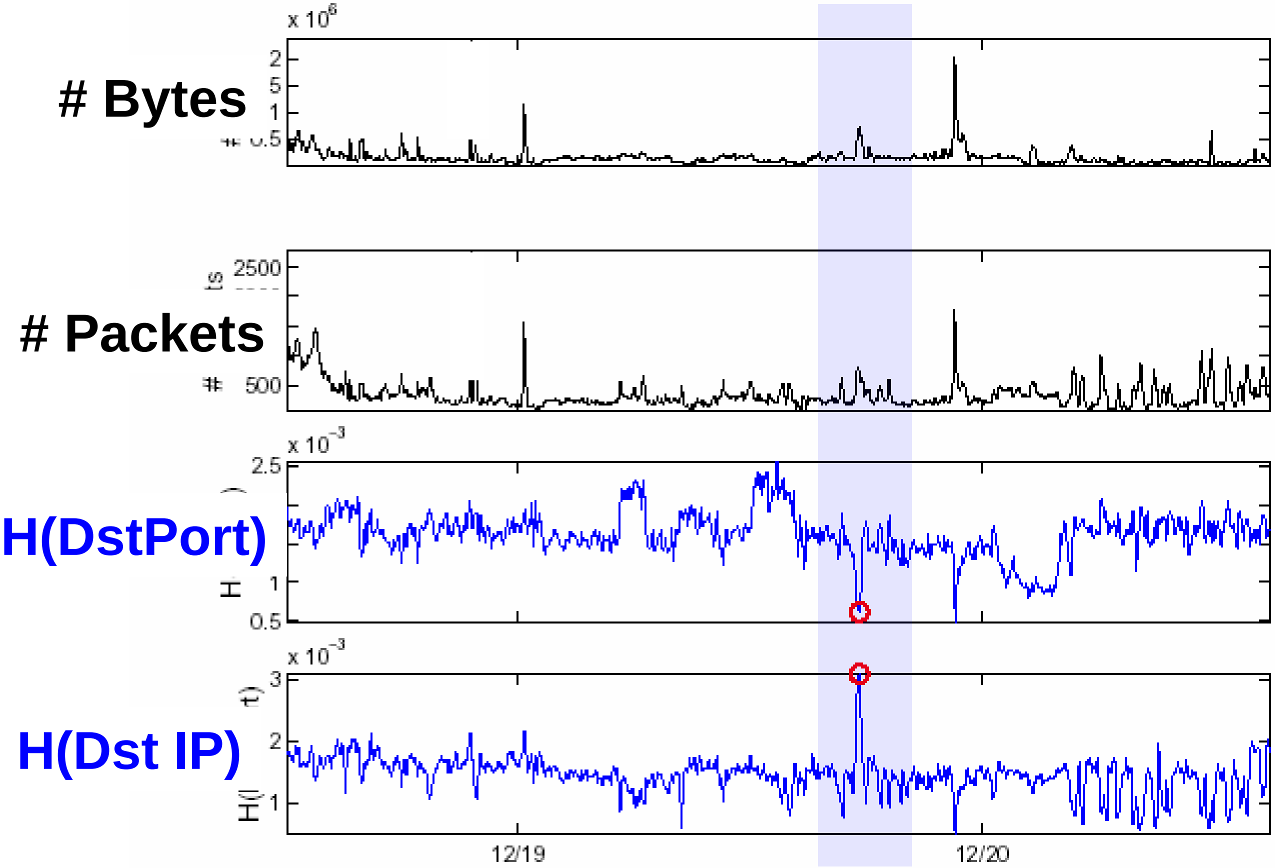
**Concentrated Histogram**
Low Entropy



**ypical Traffic**

Summarize using **sample entropy** of histogram *X*

# The Power of Entropy

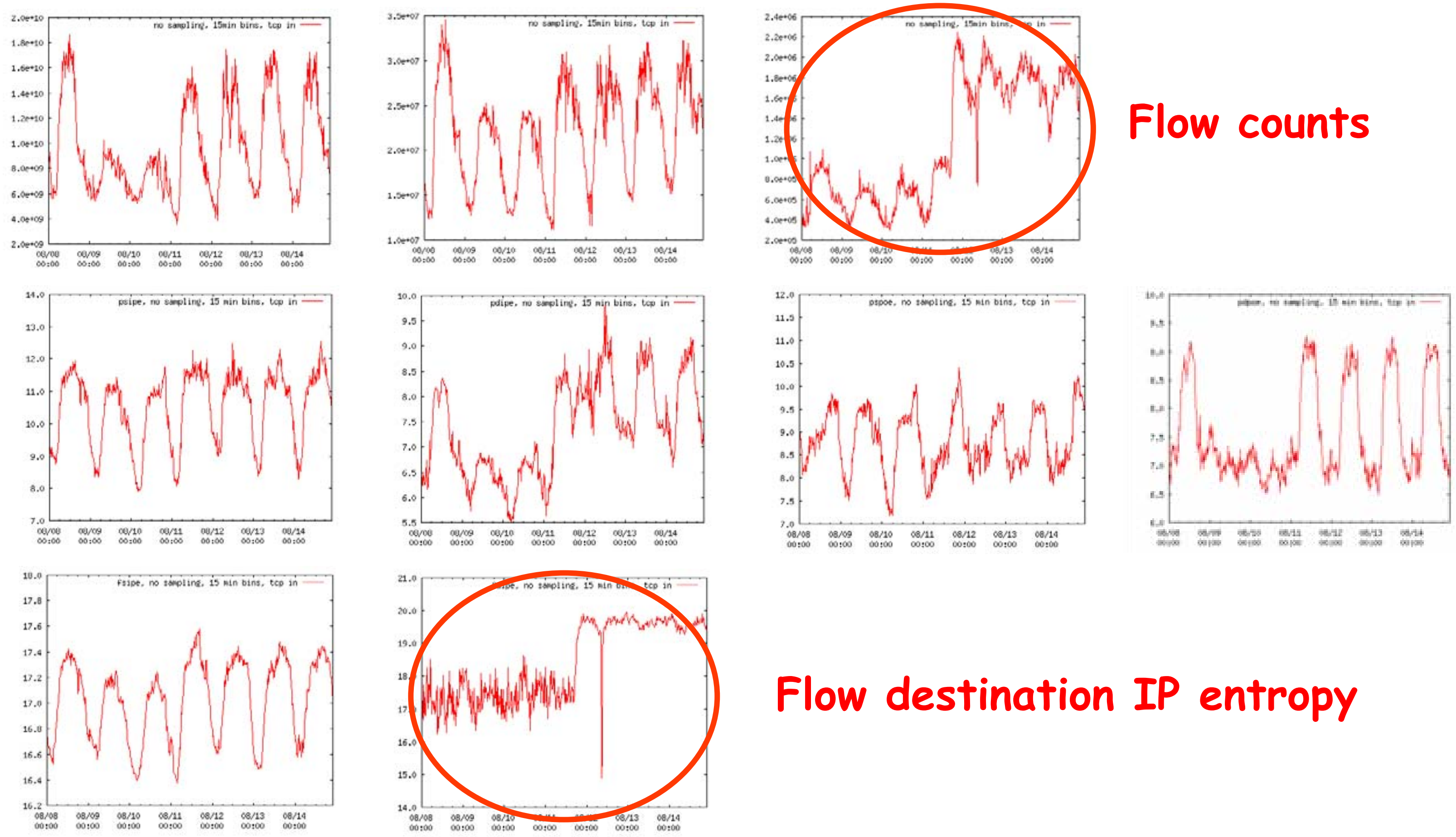


Traffic from SNVA to NYC

Worm scan dwarfed in volume metrics…

But stands out in feature entropy, which also reveals its structure

# Which AD metrics to look at?



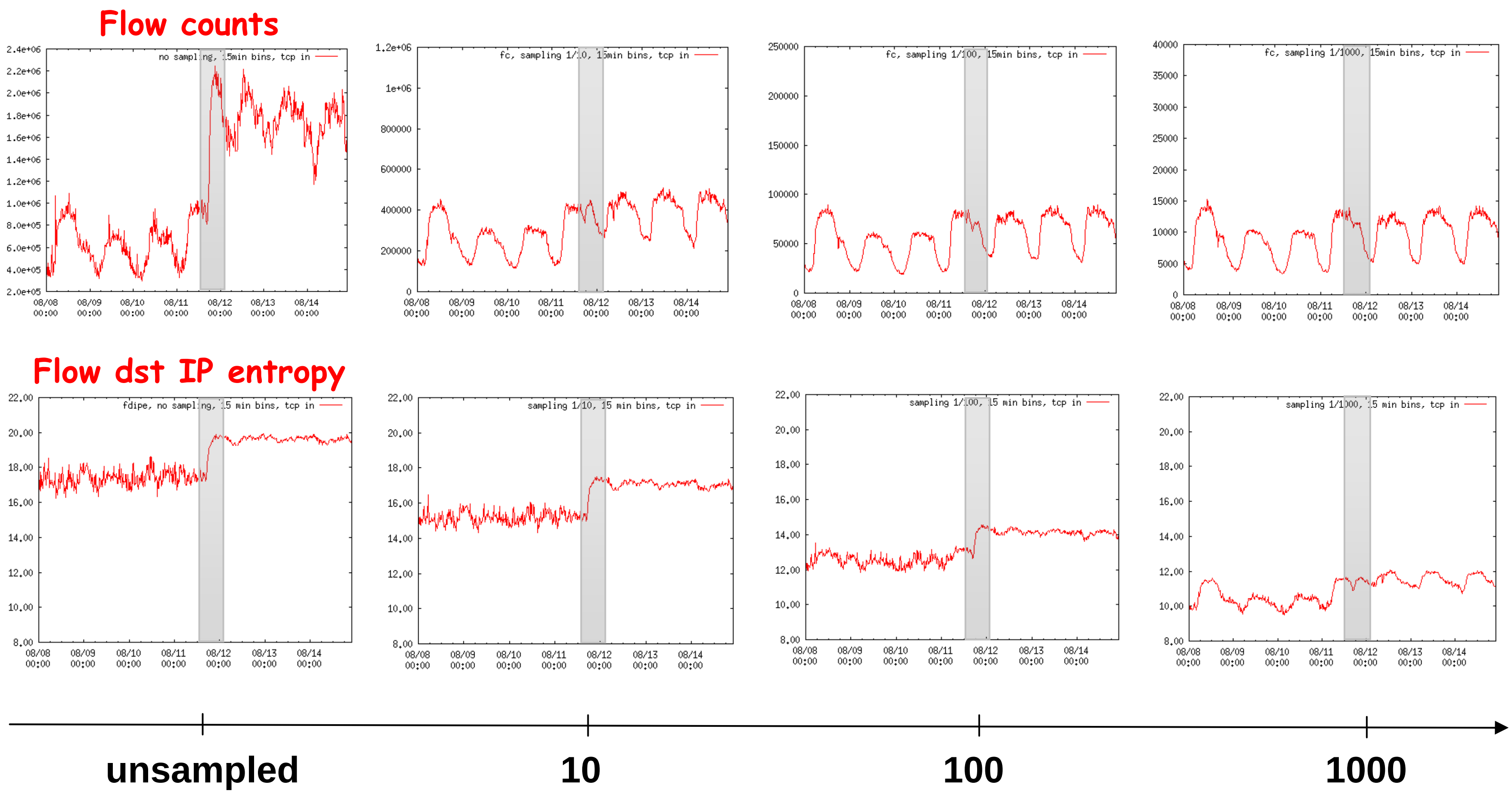
Flow counts

Flow destination IP entropy

# Methodology: Packet Sampling

- Determine the *packet size* (bytes) and *timestamps* for individual packets in the flow trace
- Each packet of a flow is recorded in it‘s own flow record with
  - *packet_size = flow_size/num_packets (average packet size)*
  - timestamp randomly chosen within flow bounds
- Randomly sample every 10th, 100th, 250th, and 1000th packet
  - Not exactly what Cisco does, but pretty close...

# Timeseries of Detection Metrics



Flow counts

Flow dst IP entropy

unsampled

10

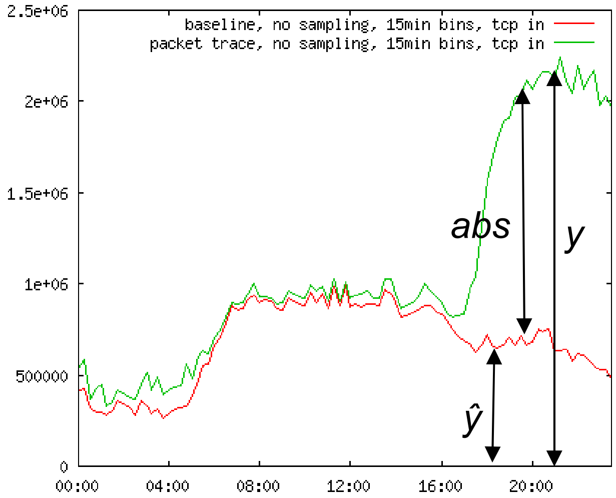
100

1000

# Methodology: Determine the Baseline

- AD algorithms measure distance from (predicted) baseline to (actual) observed metrics
- Each AD method uses it‘s own handcrafted algorithm to determine the baseline model
- Since we know the anomaly very well we can construct an *„ideal baseline“* by removing all blaster packets from the observed trace
  - *Heuristic: blaster packet = packet with destination port 135, protocol TCP, and length of 40, 44, 48 bytes*
- One baseline per metric and *sampling rate*

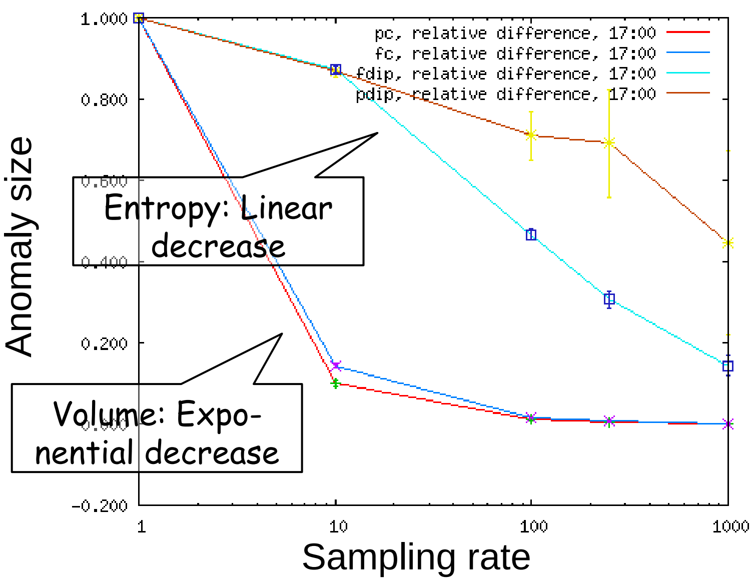
# Methodology: Measure anomaly distance



- Absolute difference between trace *y* and baseline *ŷ*
  - *abs = y – ŷ*
- Absolute difference normalized to the baseline *ŷ*
  - *rel = (y – ŷ)/ ŷ*

# Anomaly Distance vs Sampling Rate

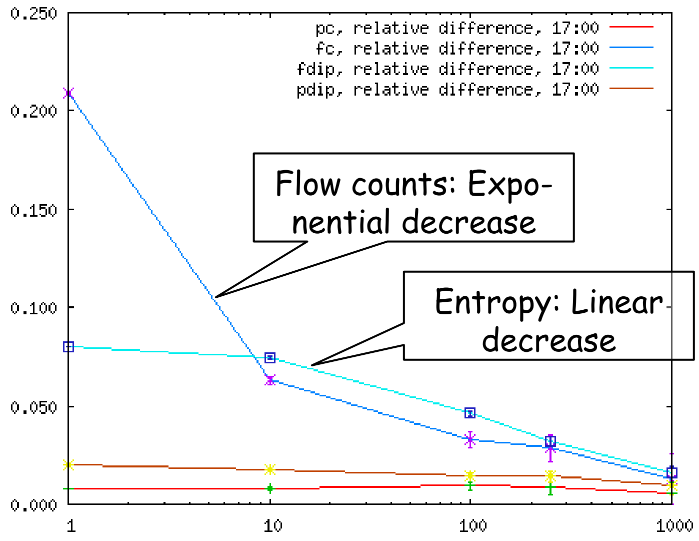
**Absolute distance**



Entropy: Linear decrease

Volume: Exponential decrease

**Relative distance**



Flow counts: Exponential decrease

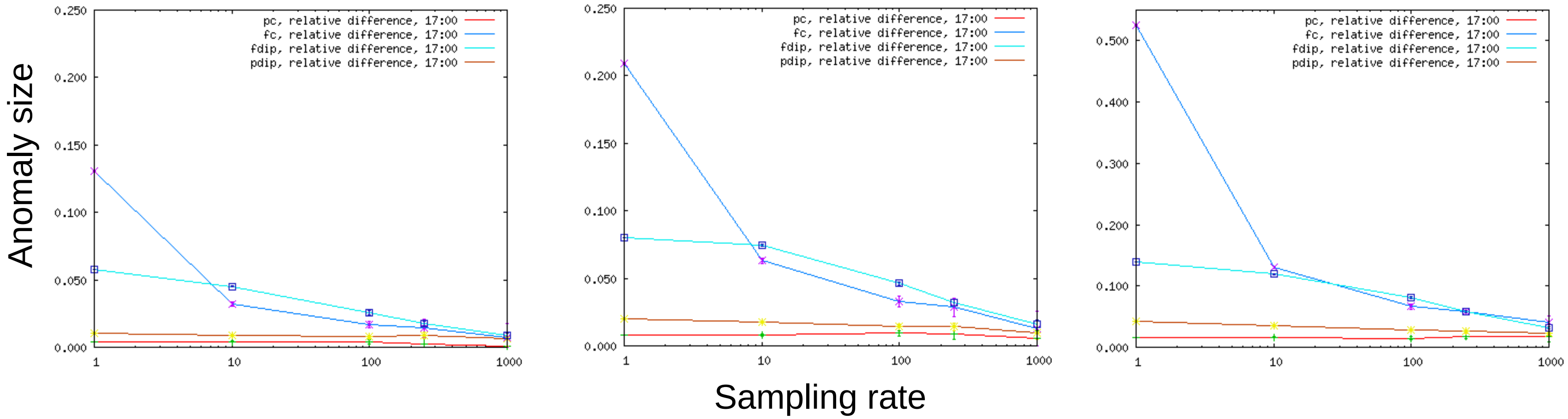
Entropy: Linear decrease

Q: What do these distance measures tell us?
A: In this scenario entropy is less disturbed by sampling…

# Scaling the Blaster Worm

- *Identification* of Blaster packets based on heuristic
  - dst port, packet size, tcp
- *Amplification* of the Blaster worm
  - Insertion of new packets with same src IP, and dst IP randomly selected from SWITCH IP range
- *Attenuation* of the Blaster worm
  - Randomly throwing out of some of the Blaster packets (e.g., select each packet with probability of 50%)

# Relative Distance for Scaled Blaster



Scaling factor: 0.5

Scaling factor: 1

Scaling factor: 2

Q: What do these scaled distance measures tell us?
A: For faster and slower Blaster-like worms, entropy is less disturbed by sampling than flow counts…

# Conclusion and Future Work

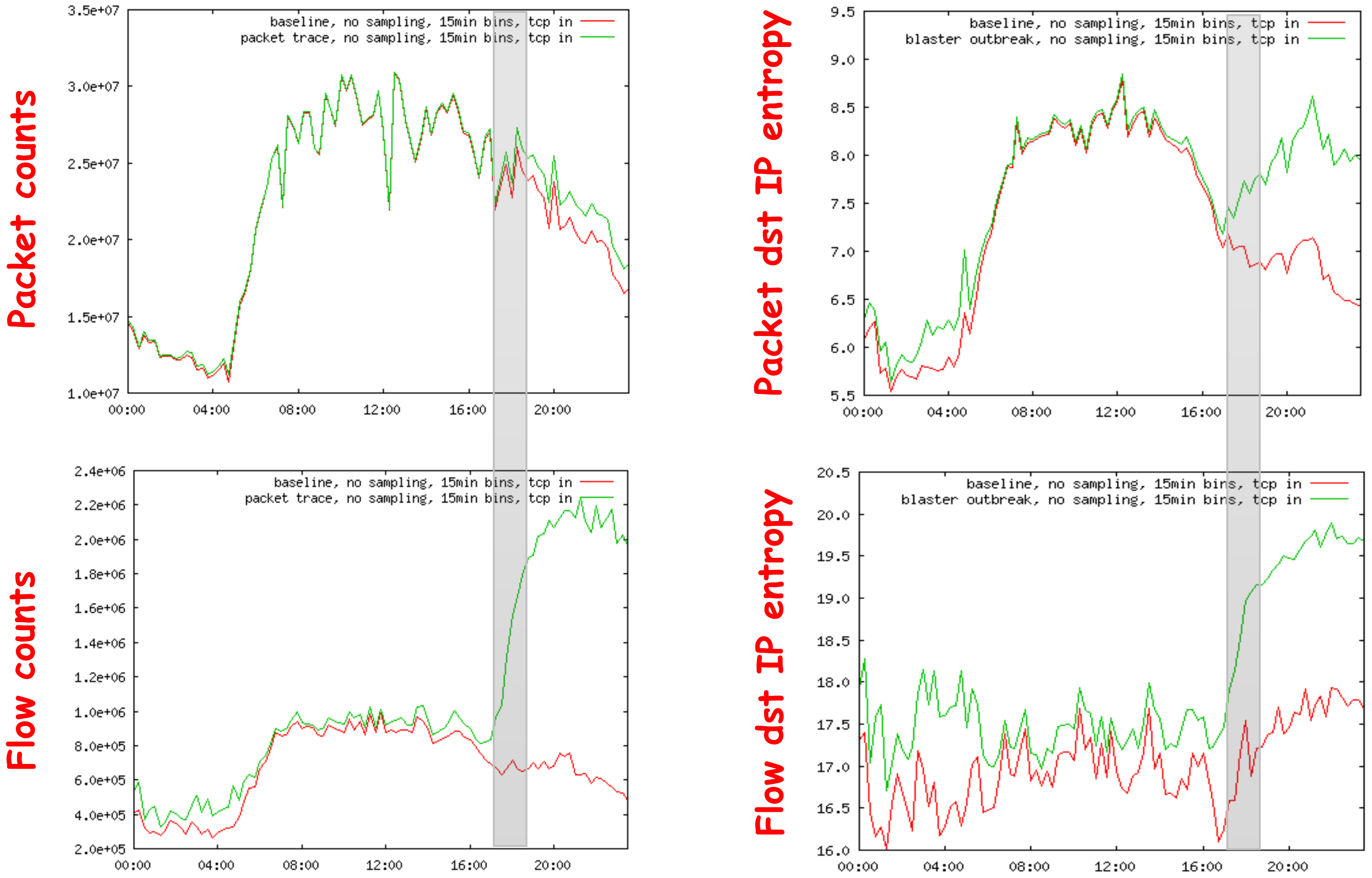
- What did we learn?
  - Some metrics are more resilient to sampling than others
  - Flow DST IP entropy is most resilient to sampling for Blaster-type anomalies (in our traces)

- What still needs to be studied...
  - Other types of anomalies, anomaly intensities
  - Other distance metrics (considering a metrics‘ variance)
  - Different bin sizes
  - Further anomaly metrics
  - Anomaly detectability at different sampling rates

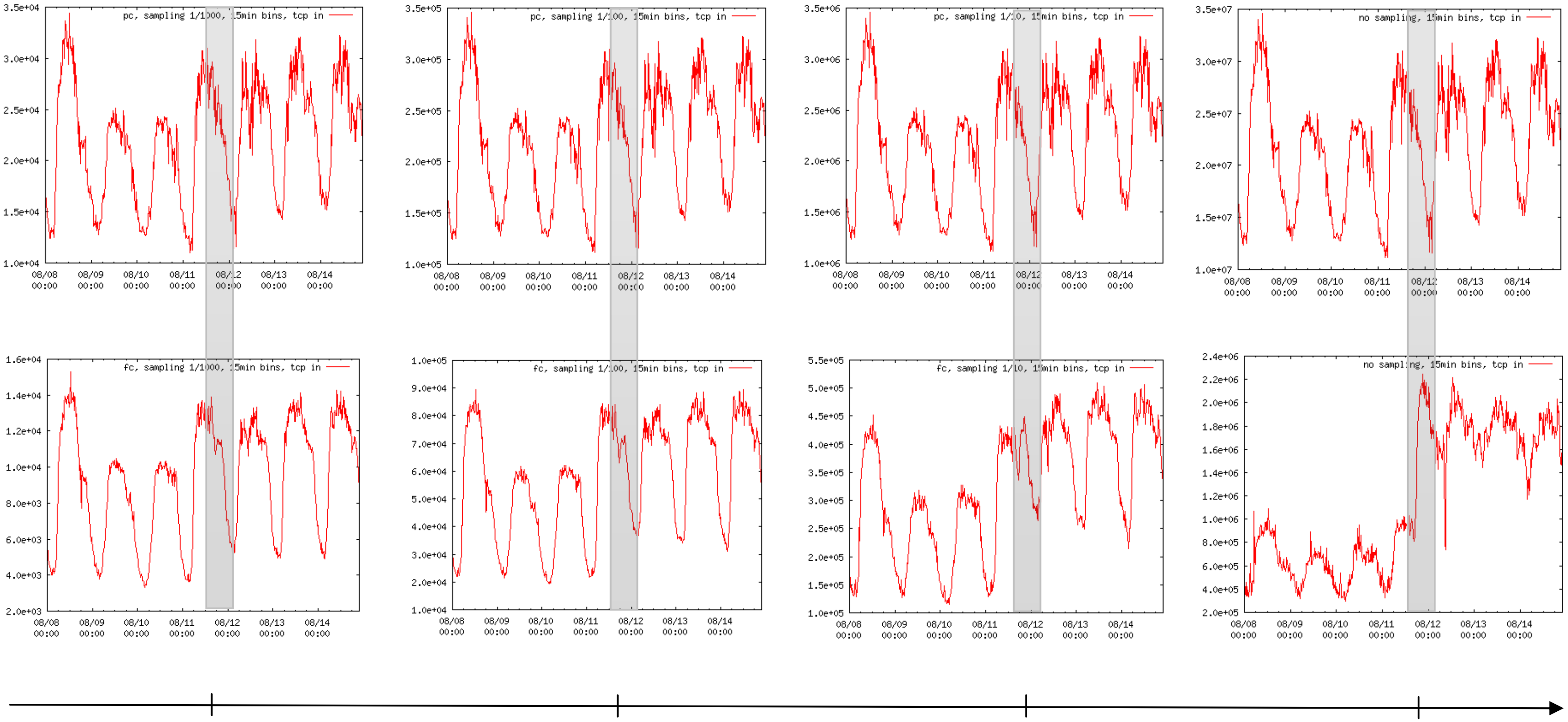
# Questions?

Daniela Brauckhoff
ETH Zurich, Switzerland
brauckhoff@tik.ee.ethz.ch

# Baselines for AD Metrics (unsampled)

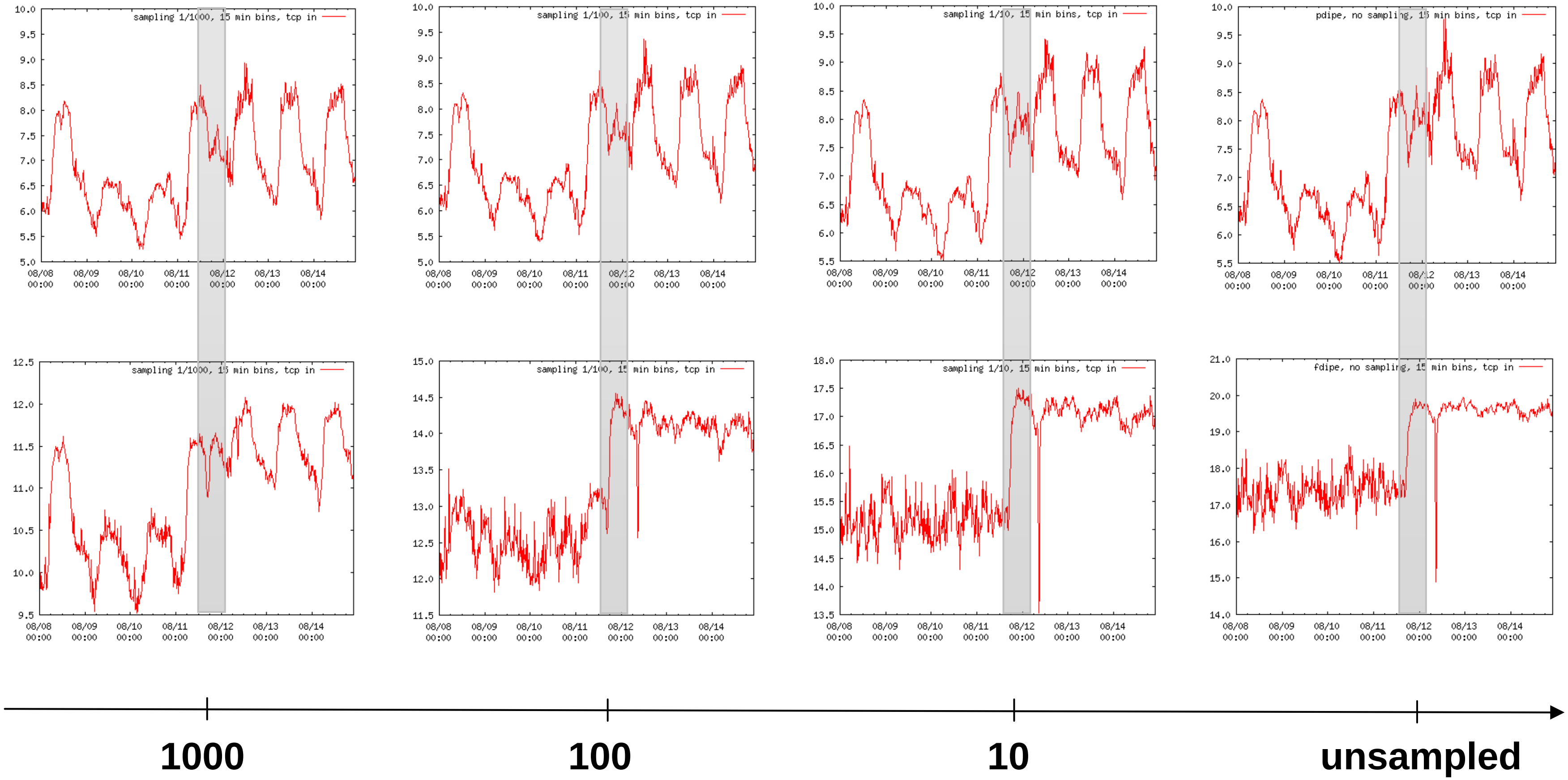


# Volume Time Series

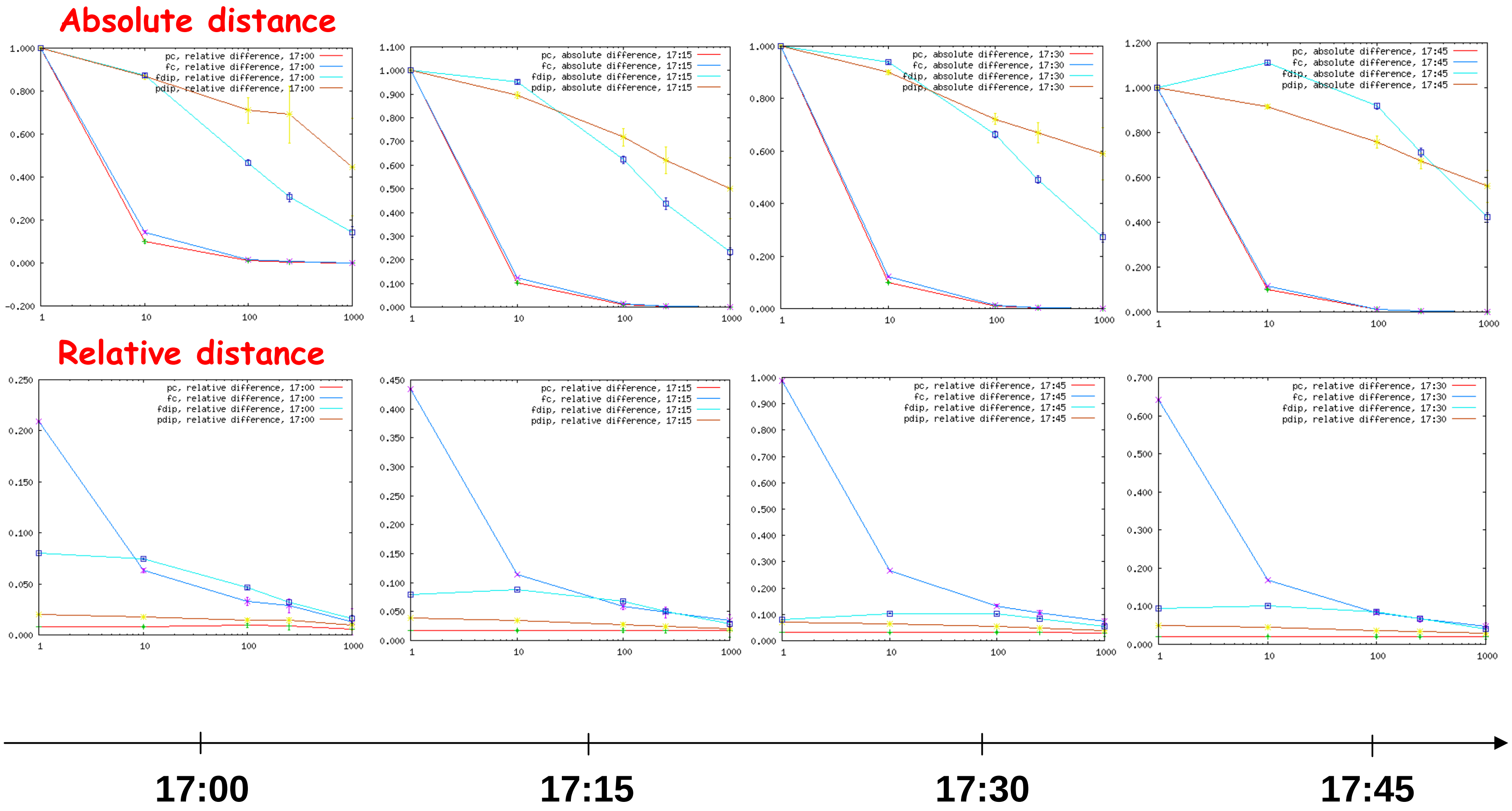


**1000** **100** **10** **unsampled**

# Entropy Time Series



# Anomaly Distance vs Sampling Rate



# A Case for Packet Sampling

FOKUS

Fraunhofer Institute for Open Communication Systems

Tanja Zseby, zseby@fokus.fhg.de

Competence Center for Autonomic Networking Technologies

# Motivation: FloCon 2005

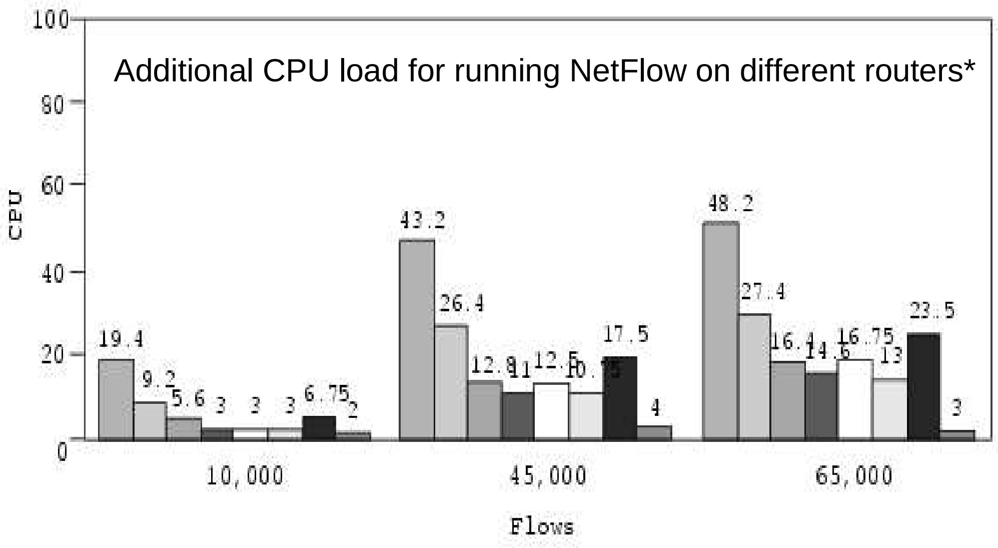
FloCon05 participants:

“We don’t believe in Sampling”

- Happy to use flow data
- Very skeptical to packet sampling

# The Problem: Limited Resources

- Full packet capture at each node not feasible
  - Increasing data rates
  - Hardware costs
  - Privacy concerns
- Resources are limited
  - Storage
  - Processing
  - Transmission



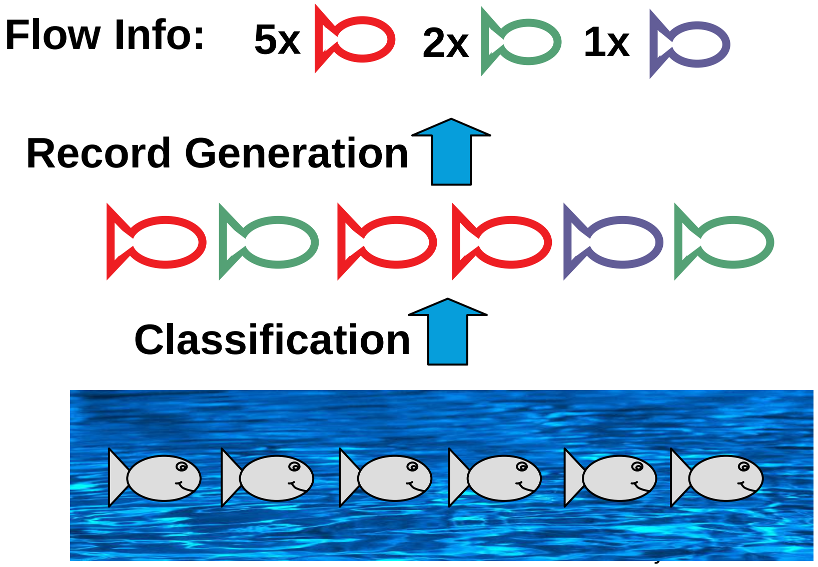
Additional CPU load for running NetFlow on different routers*

**We cannot measure everything**

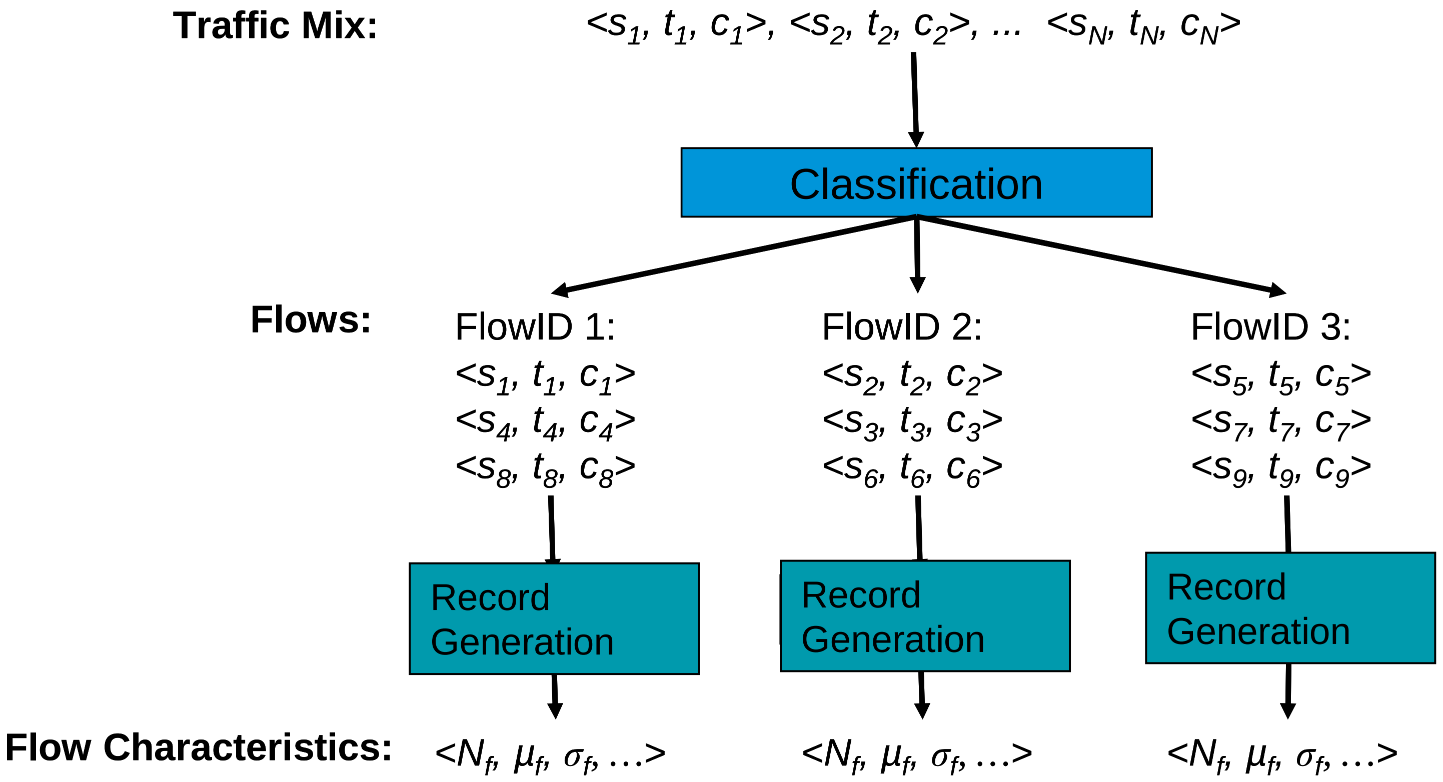
*source: NetFlow Performance Analysis, Cisco white paper
http://www.cisco.com/warp/public/cc/pd/iosw/prodlit/ntfo_wpa.jpg

# Solution1: Flow Data

- Grouping of packets into flows (classification)
- Reporting of flow information only
- Disadvantages:
  - Per-packet information is lost
  - Information and effort depends on flow definition



# Flow Data Generation



**Traffic Mix:** $<s_1, t_1, c_1>, <s_2, t_2, c_2>, \ldots\ <s_N, t_N, c_N>$

Classification

**Flows:**

FlowID 1:
$<s_1, t_1, c_1>$
$<s_4, t_4, c_4>$
$<s_8, t_8, c_8>$

FlowID 2:
$<s_2, t_2, c_2>$
$<s_3, t_3, c_3>$
$<s_6, t_6, c_6>$

FlowID 3:
$<s_5, t_5, c_5>$
$<s_7, t_7, c_7>$
$<s_9, t_9, c_9>$

Record Generation

Record Generation

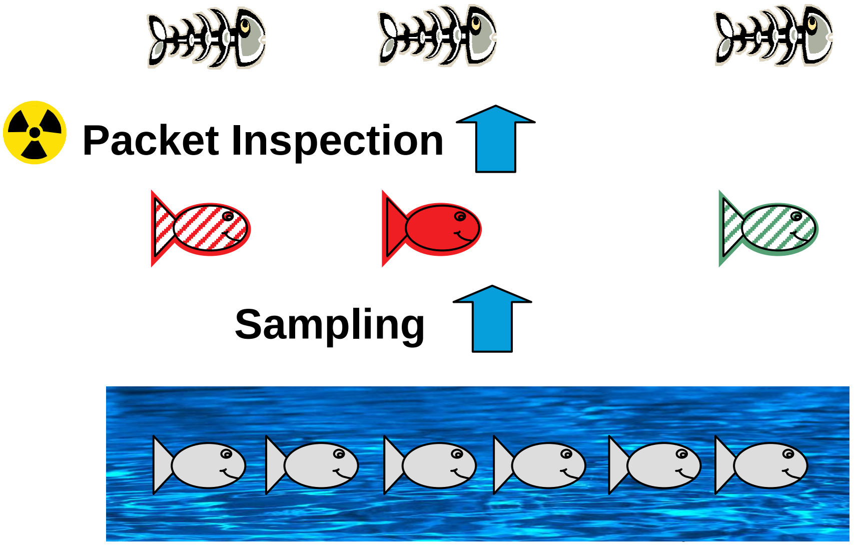
Record Generation

**Flow Characteristics:** $<N_f, \mu_f, \sigma_f, \ldots>$ $<N_f, \mu_f, \sigma_f, \ldots>$ $<N_f, \mu_f, \sigma_f, \ldots>$

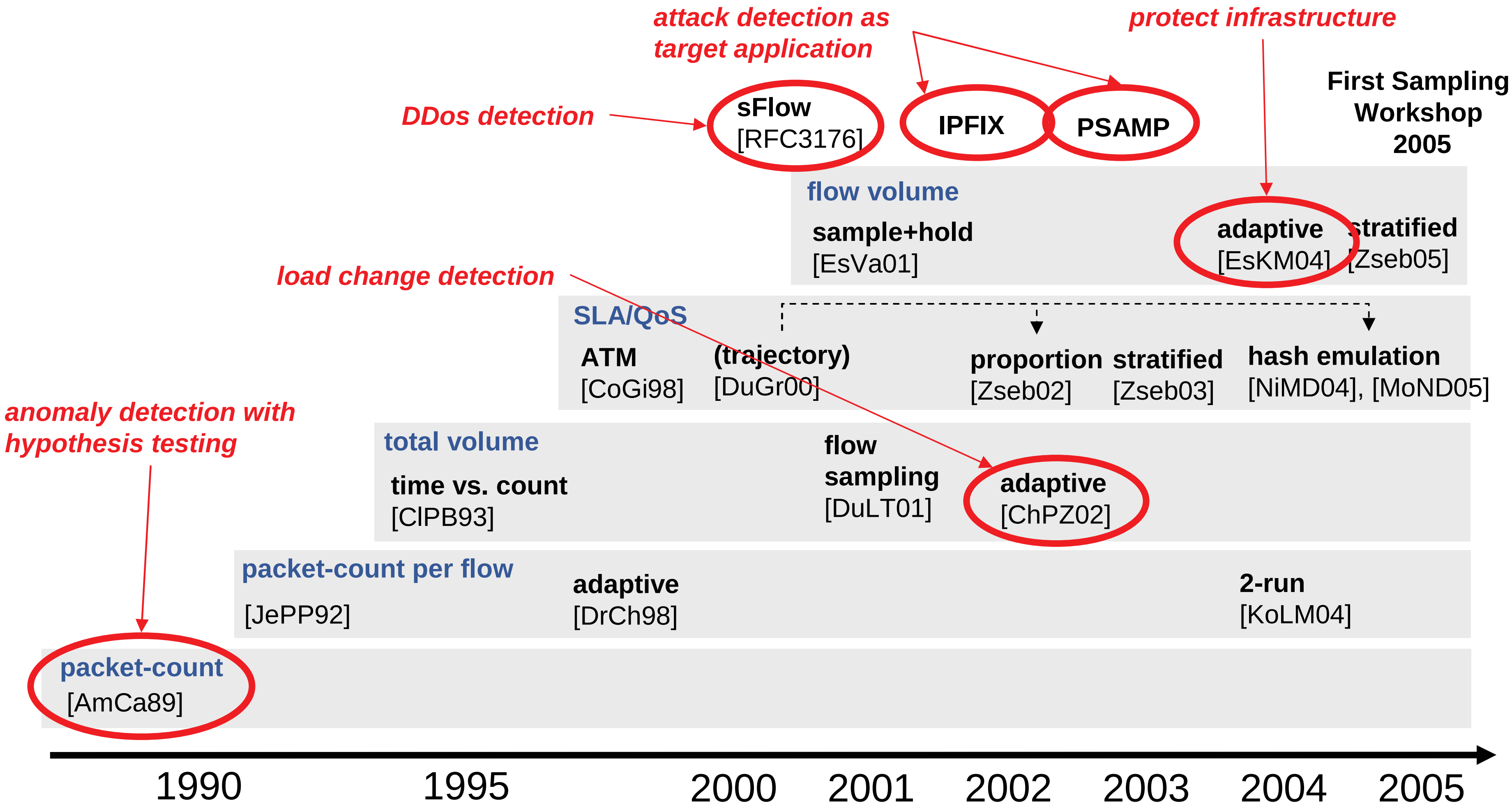
- Information about packets is discarded
- Available information depends on
  - Flow definition
  - Flow characteristics that are reported

# Solution2: Packet Sampling

- Random Selection of some packets
  - Report parts or full packet information
  - Estimation of metrics based on sample
- Provides different viewpoint
  - Packet data can reveal further information
  - Sampled data sufficient for some metrics
- Helps to protect measurement infrastructure during attack



# Sampling: State of Art



attack detection as target application

protect infrastructure

DDos detection

sFlow [RFC3176]

IPFIX

PSAMP

First Sampling Workshop 2005

flow volume

sample+hold [EsVa01]

adaptive [EsKM04]

stratified [Zseb05]

load change detection

SLA/QoS

ATM [CoGi98]

(trajectory) [DuGr00]

proportion [Zseb02]

stratified [Zseb03]

hash emulation [NiMD04], [MoND05]

anomaly detection with hypothesis testing

total volume

time vs. count [ClPB93]

flow sampling [DuLT01]

adaptive [ChPZ02]

packet-count per flow

[JePP92]

adaptive [DrCh98]

2-run [KoLM04]

packet-count [AmCa89]

1990 1995 2000 2001 2002 2003 2004 2005

# Packet Sampling

Real metric substituted by estimate

➔ Accuracy statement is essential

Accuracy depends on

- Sampling scheme
- Estimation method
- Position of sampling process in measurement sequence
- Population characteristics (e.g. variance of metric of interest)

# A Simple Example

**Goal: Estimation of packet proportions (e.g. TCP-SYN packets in a flow)**

Real proportion: $P = \frac{M}{N}$ Estimate: $\hat{P} = \frac{m}{n}$

Estimation Accuracy (random n-of-N): $\sigma_{\hat{P}} = \sqrt{\frac{P \cdot (1-P)}{n}} \cdot \sqrt{\frac{N-n}{N-1}}$

Confidence Limits: $Prob\left(\hat{P} - z_c \cdot \sigma_{\hat{P}} \leq P \leq \hat{P} + z_c \cdot \sigma_{\hat{P}}\right) = 1 - \alpha$

Example: - Measurement interval with N=10,000 packets
- Random packet selection 1% (n=100)

$\hat{P} = 0.9$ ➔ $\sigma_{\hat{P}} = 0.03$ ➔ 0.8226 ≤ P ≤ 0.977, with 99% confidence

$\hat{P} = 0.1$ ➔ same accuracy

$\hat{P} = 0.5$ (worst case) ➔ $\sigma_{\hat{P}} = 0.05$ ➔ 0.371 ≤ P ≤ 0.629, with 99% confidence

**Works with other packet properties, too!**

# Advise

- Don’t restrict your analysis to flow data
  - Include further viewpoints
  - Use sampling in addition or as alternative to flow data
- Trust the power of statistics
  - It’s a mature and well established field

  ➔ full range of proven techniques
- Use sampling where applicable
  - Applicability depends on traffic profile, metric of interest, accuracy demand
    - Sampled data sufficient to detect large events (high volumes, high packet counts)
    - May be sufficient to estimate #pkts with specific properties (e.g.  SYN, VoIP packets, small packets, packets with same content, etc.)
    - Others ➔ depends on scenario
  - Difficulties with rare events (stealth attacks, slow port scans)
  - Not suitable to re-assemble connections (but filtering may be)

# Thank you for your attention!

FOKUS

Fraunhofer Institute for Open Communication Systems

# A System Architecture for Processing Flows

Raj Srinivasan

Director, Software Engineering

Bivio Networks, Inc,

Email: raj@bivio.net

FLOCON, 9 October 2006

# System Requirements

Due to the ever increasing number of network applications, and the proliferation of threats, requirements on flow processing systems are increasing dramatically with time. The main requirements are:

- Performance – this is currently in the 1 Gbps to 10 Gbps range, and increasing
- Scalability – the system architecture must be capable of addressing the requirements at lower and higher end applications economically, by scaling the available compute resources
- Functionality – the numbers and types of applications to be supported are increasing every year. In addition to general processing capabilities, these applications require special functions to be accelerated. Common functions are:
    - cryptography
    - compression
    - regular expression matching
- Manageability – this is a key function for almost all situations, and includes a number of functions such as the ability to configure, monitor, and maintain.
- Application maintainability and portability – this is often overlooked. The architecture should be general enough so that applications can maintain generality, and address special needs through appropriate APIs. The cost of transforming applications to suit special architectures has been proven to be very costly, again and again.

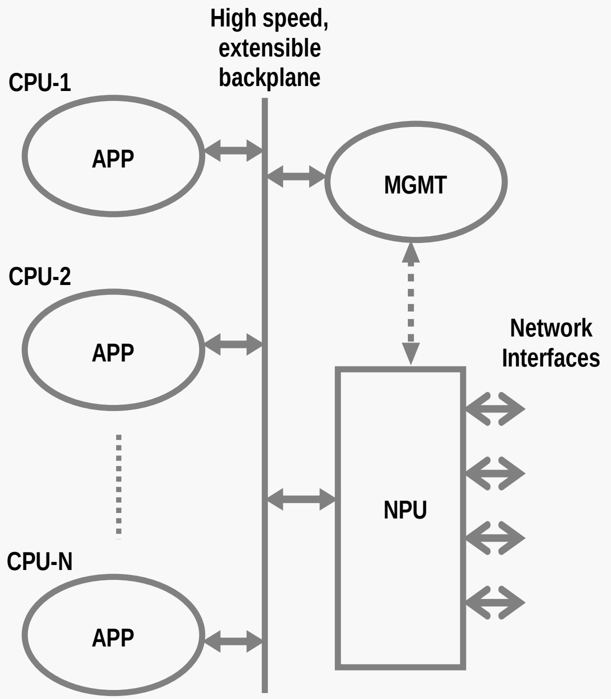
# A General Architecture that meets these requirements

We propose the following clustering architecture, and show how it meets all the requirements we have listed. Such a system has been implemented for commercial use.

This architecture has the following highlights:

- General purpose CPUs with facilities (over PCI) for specialized coprocessors
- A Network Processing Unit (NPU) for managing flow distribution
- An extensible, high-speed backplane for interconnecting CPUs and NPU
- A designated CPU for system management
- Facilities to attach multiple types of network interface cards to NPU

Architecture for Flow Processing



# The Application Environment - Software

Software Environment

- All application CPUs run the Linux operating system. They are loosely coupled, and run identical versions of the kernel and system applications.
- In the current implementation, each CPU is a PPC 7447A processor. The architecture does not depend on the processor type, though. In the next generation, each CPU is a multi-core unit that can be configured to run as single cores or in SMP mode.
- There is no shared memory between the CPUs. All communication is over the system backplane.
- Each CPU has attached to it a PCI bus. This enables to attach coprocessor/accelerator units to each CPU.
- The management CPU has storage attached to it, that is accessible to all the application processor CPUs.
- All of the system management and configuration functions – the databases, CLI, GUI – run on MGMT.

# The Application Environment – Networking, File System

Network Environment is identical in every CPU

- The backplane forms a private network which is accessed through a special Ethernet interface.
- Each external Ethernet interface is virtually extended into each CPU, so the external networks are accessed as though they were directly attached to each CPU. The NPU (Agere APP 540) is transparent.
- Applications (running in CPUs and MGMT) communicate with each other using standard internet protocols over standard socket mechanisms.
- Routing protocols run on MGMT, but forwarding tables are propagated to application CPUs.
- Interfaces are configured on MGMT, and interfaces states are propagated to application CPUs.

File system environment is identical in every CPU

- All CPUs boot over the backplane using standard protocols with MGMT serving as the boot server.
- File system is mounted from MGMT
- Each CPU has local directories so applications can have individual logs or temporary storage.

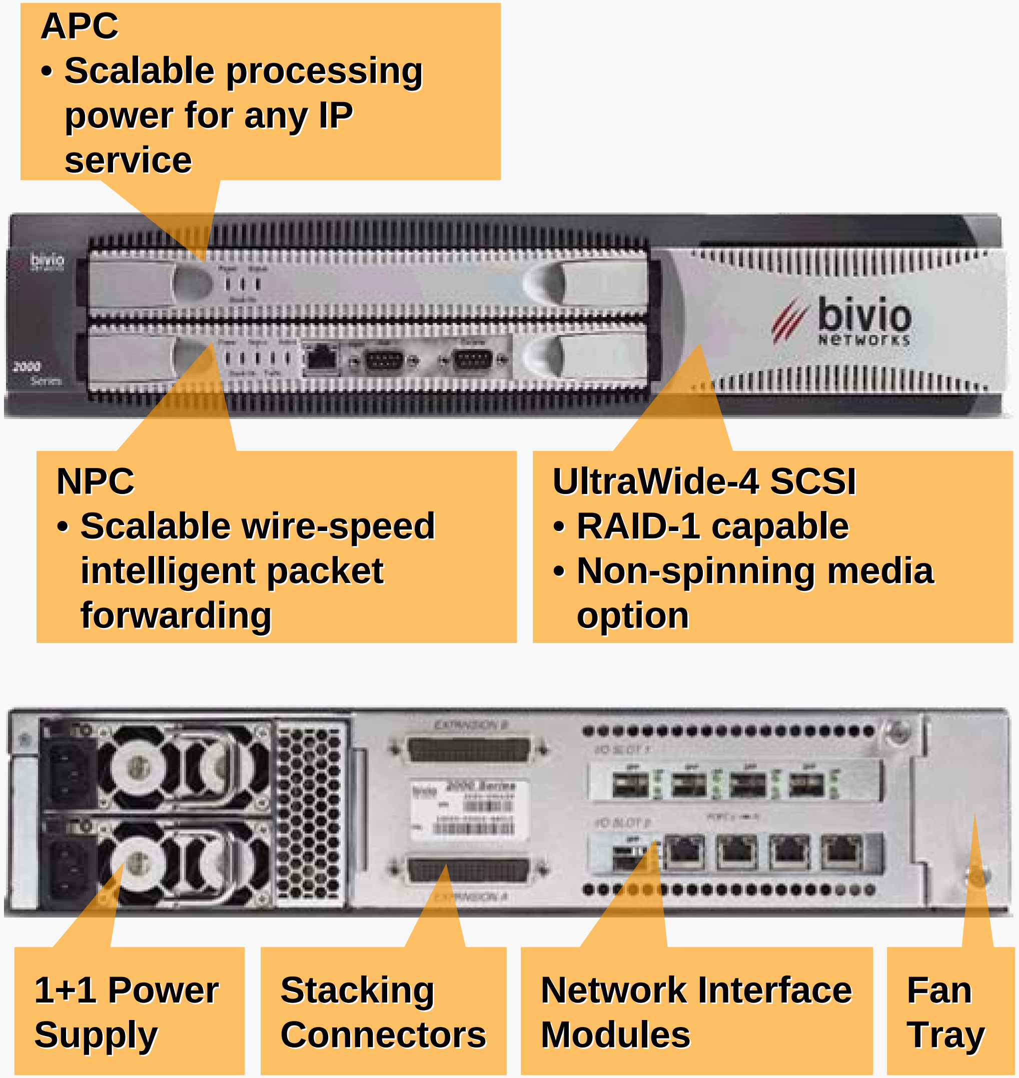
# NPU Functions

The NPU (Agere APP 540) performs the following functions:

- Load-share flows to application CPUs based on multiple (configurable) algorithms
- Support special APIs for
  - Cutting through flows that are not interesting to applications
  - Bind specified flows to specific CPUs (application instances)
- Perform QoS functions
- Maintain and report flow statistics as required by applications
- Partition traffic between different applications using the Configurable Inspection Groups (CIG) feature
- (Future) Provide hooks for third parties to insert special inspection routines to maintain application specific state. This is not possible in the current generation, but will be an integral part of the next generation system.

# Bivio Architecture Overview

- System architecture optimized for Packet Handling
- Multi-dimensional scaling
- Key hardware elements:
  - Powerful computation platform
  - Hardware acceleration
- Network computing platform
  - Integrated management
  - Extensive QoS features
  - Built-in high availability options
  - Architectural survivability
- Open development environment
  - Linux execution environment
  - Full featured SDK



APC
- Scalable processing power for any IP service

NPC
- Scalable wire-speed intelligent packet forwarding

UltraWide-4 SCSI
- RAID-1 capable
- Non-spinning media option

1+1 Power Supply

Stacking Connectors

Network Interface Modules

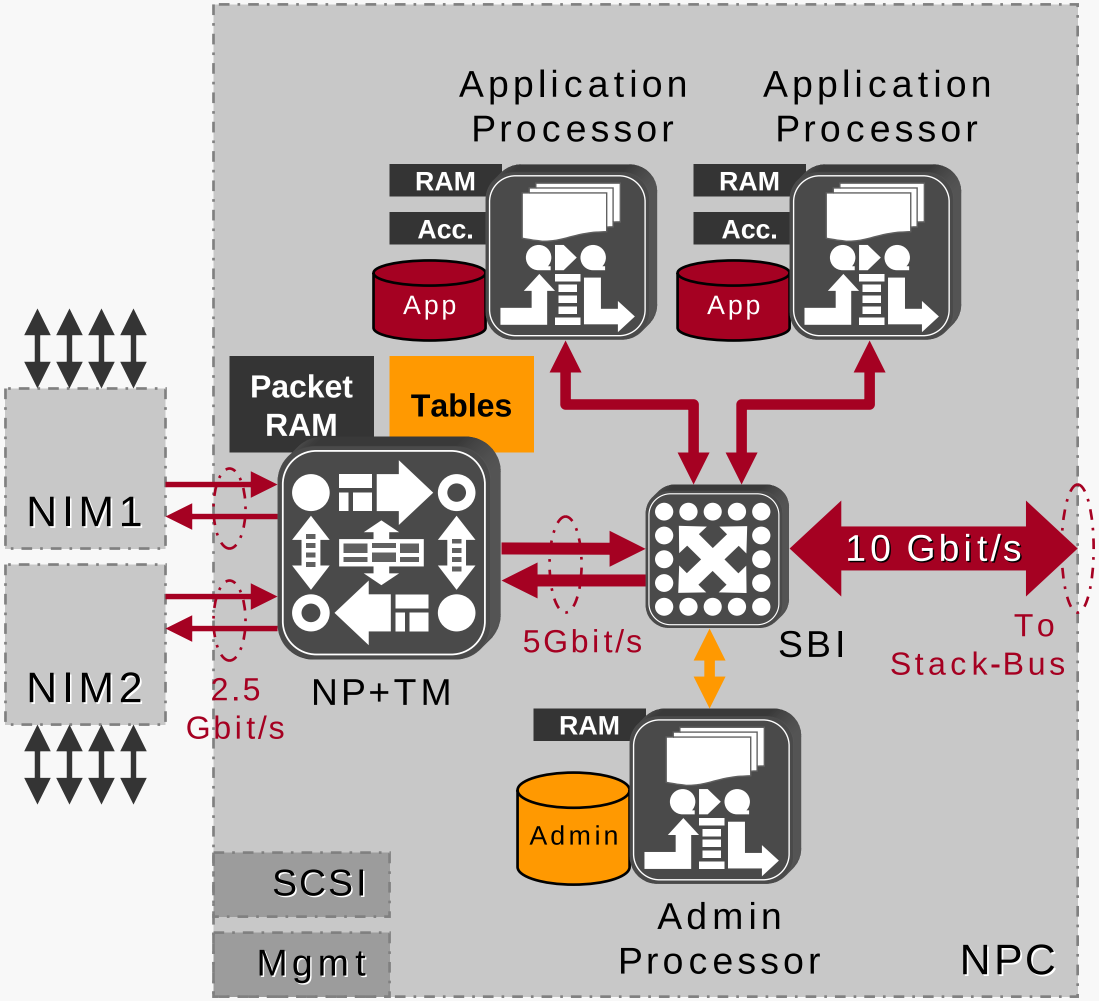
Fan Tray

# NPC: Network Processor Card

- 3 Gbps aggregate throughput
- Wire speed intelligent forwarding operations
  - Packet inspection & classification
  - Modification & transformation (NAT)
  - Traffic Management
    - Diffserv, Intserv ..
- CPU load sharing control
- Internal Dual SCSI HD and RAID support
- Chassis-level management
- DB-9 and 10/100 RJ-45 management ports

# NPC Functional Overview

- PowerPC-based Application and Admin Processors
  - ~6,000 MIPS on local APs for application processing
- Network Processor and Traffic Management subsystem
  - 5Gbit/s full duplex performance
  - QoS processing

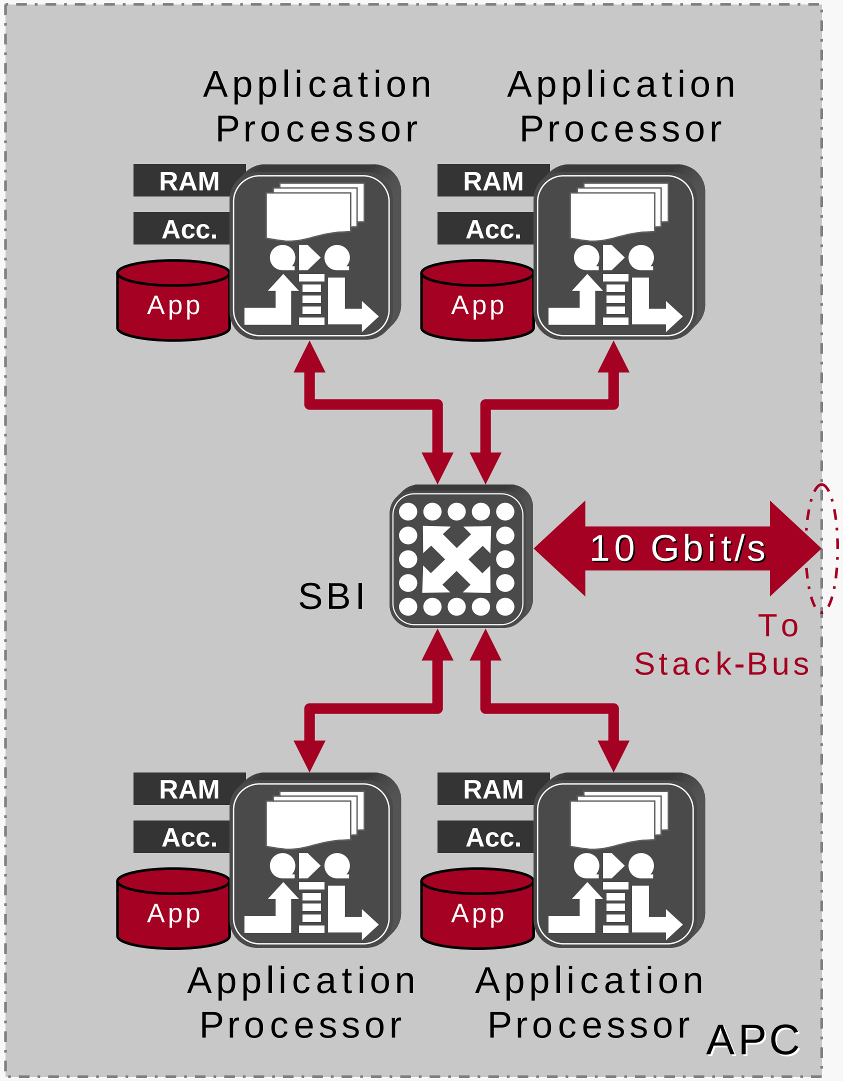


# APC: Application Processor Card

- Fully redundant computing architecture
- Parallel Motorola PowerPC subsystems
- Modular hardware acceleration per CPU
- Standard Linux execution environment
- Flows distributed across CPU cluster by NPC

# APC Functional Overview

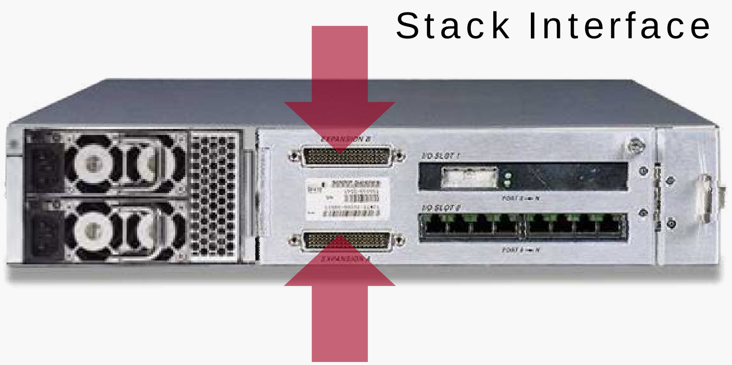
- PowerPC-based Application
- Any number of APCs can be stacked

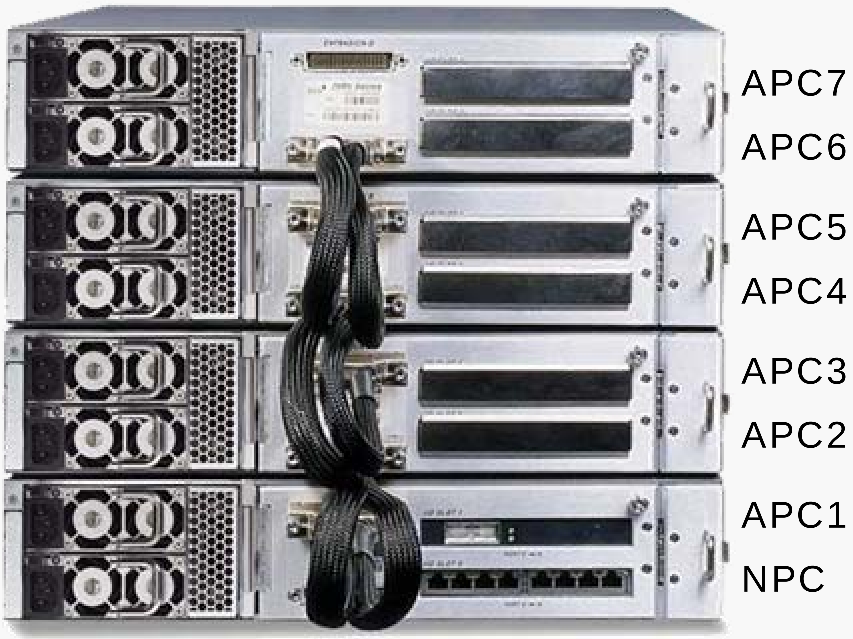


# Scaling

- 10 Gbps full duplex stack interface
- Multi-dimensional scaling
  - Additional APCs scale processing
  - Additional NPCs scale throughput
- Uniquely suited to demands of current and future Packet Handling applications

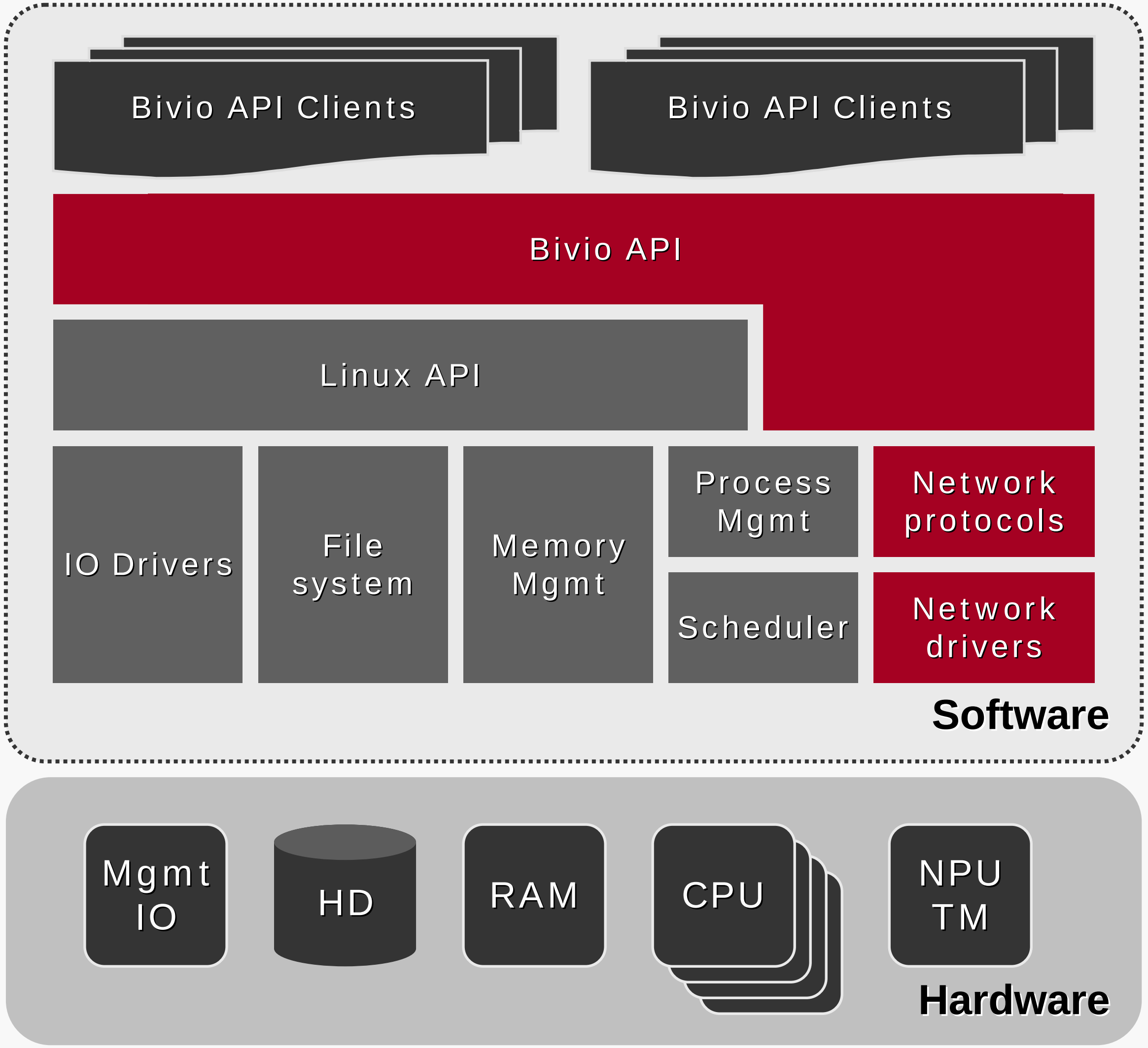
**US Patent Application #10/078,324**
**“Systems and methods for fair arbitration between multiple request signals”**





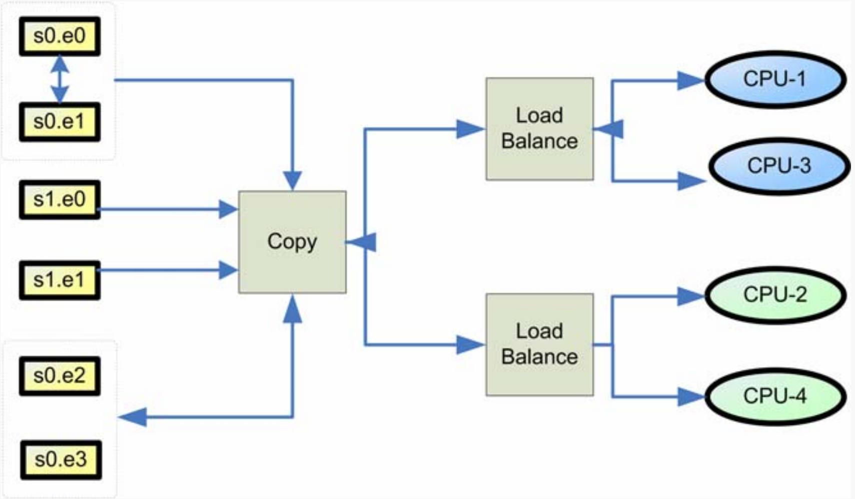
# Open Software Architecture

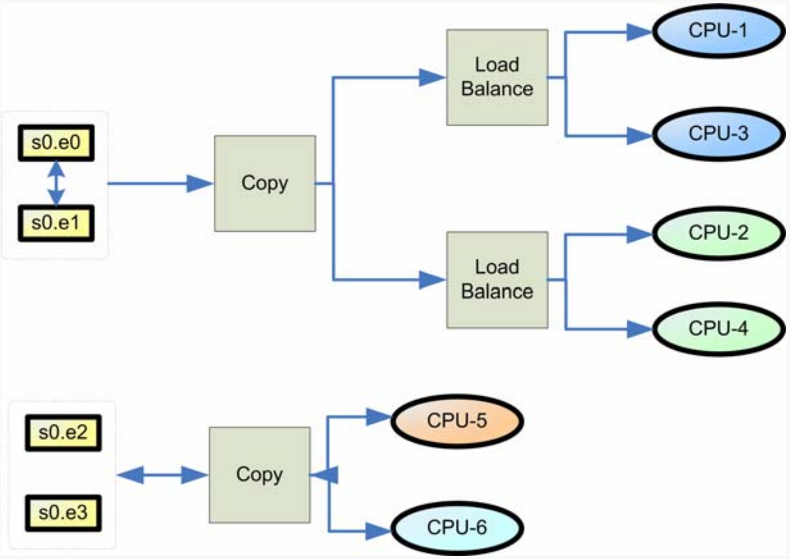
- Standard Linux execution environment, enhanced with:
  - Performance booster features such as **ZeroCopy** drivers
  - Congestion control mechanisms
  - Configurable Inspection Group feature
- SDK enables applications to leverage platform features
  - FW/VPN
  - IPS/IDS
  - Load balancing
  - Managed services
  - UTM
  - Any proxy applications
- Application porting support



# Configurable Inspection Groups

- Advanced Modes
  - Multiple Inspection Groups
  - Copy mode
  - Tap mode
  - Load balancing





# Performance & Scalability

Until recently, Moore's law ensured that performance goals could always be met using more and more powerful processing units. But this has changed recently, as evidenced by the move by major processor vendors to multi-core chips. However, this too has some problems:

- SMP units are bound by memory bandwidth. Even with two memory controllers, the classic architecture of a powerful dual-core SMP CPU operating on very fast memory (600+ Mhz DDR2 memory) will be hard pressed to do a 1-2 Gbps of flow processing. There is no way to get to 10 Gpbs with such an architecture, even with multiple cores, operating on a single memory subsystem.
- ASICs are limited by the same issues. Ultimately, an ASIC incorporates some sort of a special purpose CPU operating on special memory. For extremely special purposes, high performance can be obtained, at the cost of flexibility and maintainability of software. Even these have to be clustered for 10 Gbps and beyond.
- When multiple applications need to be working in a cooperative environment, such as in a Unified Threat Management System (UTM), each with differing requirements on processing and memory resources, a cleaner separation than afforded by an SMP is sometimes preferable. Virtual architectures provide the separation, but are often performance limited due to the overheads.

One of the great advantages of the Clustering architecture we have presented is the inherent linear scalability. There is also a lot of built-in redundancy, since CPUs can be used interchangeably.

# Scalability - continued

The NPU makes a big contribution to system scalability, if used appropriately.

NPUs are typically good at traffic processing, and a certain level of inspection. In our architecture, we do not depend on NPUs to perform special functions such as RegEx or Cryptographic processing. These are performed on CPUs, and so can be scaled.

Within NPUs, we do the following:

- Intelligent load sharing, to ensure that both directions of a flow are processed in the same CPU. This cuts out a lot of synchronization traffic between CPUs.
- (Future) Deep packet inspection to ensure that related flows will stay on the same CPUs. A simple example is the SIP signaling and media sessions, or the FTP control and data sessions. This also reduces synchronization traffic drastically.
- Configure special computing groups for special traffic classes. This feature organizes the available computing resources into groups that process special (specified) traffic classes.
- Recognize failed CPUs and redistribute traffic to other compute resources.

Doing these, we have found that performance and functionality can be scaled LINEARLY by adding more CPUs and NPUs. We believe that the biggest cause of less than linear scaling is synchronization traffic, and reducing this is one the NPUs very important contributions to this architecture.

# System Manageability

This is such an important feature that we have dedicated a CPU for management. This has the following benefits:

- The system is always accessible, configurable, and manageable under the heaviest load conditions. The importance of this property cannot be overemphasized since there are many situations that need to be addressed immediately, and delays cannot be tolerated.
- The application CPUs are spared some of the more expensive management functions involving log processing, configuration updates, etc.
- MGMT in our architecture does some special functions such as system health monitoring, and upon detecting critical failures, can trigger failovers with very small latency.
- Many system maintenance functions can be done on the fly.

# Summary & Ongoing Work

In summary, we have presented a simple architecture that has the following features:

- Scalable for CPU performance
- Scalable for specialized coprocessor functions
- Scalable NPU capabilities
- Linearly scalable by choosing the appropriate mode of operation for NPU
- Standard application environment that promotes ease of maintenance and application portability
- Manageability

In addition, this architecture has been implemented for commercial applications, in a highly integrated 2U chassis.

The next version of this product is under development, and will incorporate more powerful processors, utilizing the best of SMP and Clustering features in CPUs, faster backplane, and incorporate an NPU that can support 10 Gbps operation.

# RAVE : The Retrospective Analysis and Visualization Engine

Phil Groce
*CERT Network Situational Awareness Group*

John Prevost
*CERT Network Situational Awareness Group*

## Abstract

As tools for collecting flow data and other network metric information improve, we must more often consider how to present that data to end users for analysis. Existing approaches tightly couple generation of analytical products with the presentation of those products. Unfortunately, this tight coupling forces tradeoffs between analytical power and interface usability.

The Retrospective Analysis and Visualization Environment (RAVE) provides data analysis and visualization capabilities independent of user interface. Applications may interact with RAVE analyses directly: using RAVE as a software library acting on local data, or remotely: communicating with a central server over the network. In both configurations, RAVE caches intermediate and final analytical products for use by multiple applications.

In this paper, we present RAVE as an analysis service provider. We discuss problems we encountered implementing RAVE and a web-based network monitoring interface that uses it. Finally, we identify places where RAVE can be improved and expanded, and ideas for possible enhancements that require further evaluation.

# Scalable flow analysis

Abhishek Kumar Sapan Bhatia

*Abstract*

While current toolkits for analysis of flow-records such as SiLK are powerful and versatile, real-time analysis of flow records at very large flow collection installations continues to be a challenge. In this paper we present a new approach for summarization and analysis of flow records. Through the use of approximate data structures, a large bulk of flow records is reduced to a compact representation that is 100 times smaller in volume than the original flow records. The techniques are suitable for implenting a small number of predefined queries that are evaluated repeatedly in a periodic manner. The operations involved in summarization and query processing are fast enough to keep up with 2.5 million flows per second in a software implementation running on general purpose hardware.

## I. Introduction

Collecting flow records is the dominant method for retaining state about the traffic observed at various links in a network. Large flow collection installations primarily face bottlenecks of four types. First, the generation of flow records from the packet stream is a challenging task, especially at the higher end of link speeds. Second, the transmission of the flow records to a central operations center is problematic due to the large (and unpredictable) volume of flow records. The third issue is that of the vast amounts of storage and its management required in large flow collection installations. Finally, analyzing a large number of flow records has a high computational overhead that often grows super-linearly with the number of flow records being analyzed together.

In this paper, we present a new approach to flow analysis. Instead of providing deterministic answers that are guaranteed to be precise and accurate, this approach uses probabilistic techniques and provides approximate answers to most queries. But relaxing the requirement for deterministic accuracy in the favor of probabilistic guarantees allows for a solution that is significantly faster and more efficient in its communication and storage requirements.

The approach presented in this paper is not meant to be a replacement for deterministic flow analysis tools such as SiLK [1], [2], [3]. Instead, we believe the two to be complementary . Deterministic analysis provides precise and unambiguous solutions, and provides a high degree of flexibility in the definition of the queries, albeit at high computational, storage and transmission overheads. The approach presented here trades off the deterministic guarantees for efficiency gains that allow such analysis to be performed at scales that are two to three orders of magnitude larger. The improved scalability of this approach implies that it provides an excellent means to perform routine analysis for a high volume of flow records, identifying a small number of suspicious cases that merit further investigation. It is also a suitable candidate to build situational awareness systems that evaluate a fixed set of queries over the flow data in a periodic fashion.

In the next section we provide a taxonomy of queries posed during flow analysis. Section III looks at an example query and provides a solution using the proposed approach. Section IV generalizes this solution to present a comprehensive summarization and analysis system for automated, routine analysis of flow data. We discuss the advantages and limitations of this approach in section V and conclude with pointers to future work in section VI.

## II. Taxonomy of Queries

We divide the universe of queries that can be posed against flow records into three classes. First, there are the *aggregate queries*, that refer to various totals among the dataset in question. Typical examples of aggregate queries are estimating the total number of flows, the total number of sources by IP, the total number of destinations identified by IP alone or by `<destination IP, destination port, protocol>` tuples. Such aggregate quantities are the broadest and most important indicators capturing the details of network activity.

The second family of queries covers *distributional queries*, pertaining to questions such as “what is number of sources that have contacted exactly one destination within my network?” or “what fraction flows have less than 10 packets?”. The broadest distributional queries ask for the entire observed distribution of a metric, such as the distribution of flow sizes, or the distribution of number of destinations contacted by various sources. Answers to distributional queries are more precise and more sensitive indicators of various kinds of network activity and attacks. For example, a small DoS attack on a web-server might not show up as a large increase in the total number of flows, but it will cause a more significant change in the number of single-packet flows.

Finally, *identity queries* try to identify specific entities that are outliers according to some metric evaluated over the dataset. For example, “which sources have contacted more than a hundred unique destinations” or “which destinations have received more than two thousand flows” are queries that identify individuals that exceed respective thresholds for the number of unique destinations and the total number of flows.

In theory, there is an immense number of possible queries that can be posed against a given collection of flow records. But in practice, there are two kinds of queries that are actually evaluated. The first set is that of *routine queries* that are evaluated regularly, in a periodic manner, over the flow records collected in regular monitoring intervals. Typically this set includes a small number of queries of all three types described above. The results to these queries can be used to visualize the

| Number of Sources | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|
| Execution Time (sec) | 9 | 50 | 581 | 922 | 1573 | 2464 |

TABLE I
TIME TAKEN TO EXECUTE AN EXAMPLE SILK QUERY OVER 512K ($2^{19}$) FLOW RECORDS.

current status of the network traffic, or in other words, provide situational awareness about the network traffic. Anomaly detection systems can track the results of such queries to detect and flag significant deviations from the norm, perhaps raising an alarm for an analyst to evaluate additional, non-routine queries. This second set represents what we call *forensic queries*. Such queries are typically part of an investigation prompted by specific network events or perhaps even an alarm raised on the basis of the information provided by the routine queries. The range and objective of forensic queries is significantly larger than that of routine queries. In this paper, we propose a more efficient approach for evaluating routine queries, keeping in mind that such queries are evaluated repeatedly and over all data that is collected. Forensic queries on the other hand, are evaluated less frequently , and are better implemented using traditional flow analysis tools such as the SiLKtools that can support a range of complex queries, albeit with significantly higher costs in computational, storage and transmission complexity. We now provide an example query and its probabilistic implementation. We will generalize from that implementation to present a broader system for routine analysis of flow data.

## III. AN EXAMPLE QUERY

Consider a query for identifying sources that have contacted an unusually large number of destinations within the monitored network. For a large network, evaluating such a query periodically over the flow records corresponding to the inbound traffic can expose a variety of activity such as port scans, flooding attacks and automated worm or botnet spreading attempts. Tracking the set of heavy hitters in this query, i.e., the sources sending the largest number of flows into the network, and changes in this set, can yield a list of suspects warranting further investigation.

Using SiLK, an implementation of this query might look like:

```
%rwuniq --distinct-destinations data.raw
```

As demonstrated in Table I, the total running time for this query increases with the total number of sources. More significant is the fact that all the half million flow records in this example need to be available at the analysis station for this query to be evaluated. Also, the amount of memory required to implement this query and the total number of memory accesses both increase no slower than the total number of flows, the total number of sources, and the number of distinct destinations per source. In the event of an actual attack or scan, the situation is further exacerbated due to the increase in the number of distinct sources and/or destinations, often by several orders of magnitude, caused by such events.

We now describe a probabilistic solution implementing this query. Consider a simple array of counters, indexed by a hash function, such that the range of the hash function equals the number of counters in the array. Now for every flow record, a hash of the source IP can be used as an index into the counter array, to locate the corresponding counter. This counter can be incremented for every flow, thus accumulating the count of the total number of flows from the corresponding source IP. Sources sending a large number of flows can be identified by selecting the flows that cause any counter to reach a predetermined threshold value. This solution is approximate in the sense that collisions in hashing can result in counter values being incremented due to more than one source IP hashing to the same location. However, with a reasonably large array, say with $2^{18}$ counters, the probability of collisions among two large sources is quite small. Such data structures, also known as *sketches*, have been studied recently for their suitability in estimating various statistics about network traffic. We refer the reader to [4], [5] for a detailed treatment of the accuracy of estimation using such sketches.

This solution can track the total number of flows corresponding to each source, but requires another component to track the set of unique destinations contacted by each source. We implement this functionality by adding a Bloom Filter to estimate whether a `<source IP, destination IP>` tuple is unique. Bloom Filters [6] are probabilistic data structures capable of answering set membership queries in very efficient manner. To count the number of unique destinations contacted by each source in an approximate manner, a second counter array is maintained, but updated only for previously unseen `<source IP, destination IP>` tuples.

Figure 1 depicts the overall solution, consisting of two counter arrays and a Bloom Filter for determining new unique destinations for a given source. The first array provides the total number of flows per source and the second array provides the total number of unique destinations per source. The estimation techniques developed by Kumar et al. [4], enable the computation of the entire distribution of the number of flows sent by individual sources, and the number of unique destinations contacted by individual sources. This answers all distributional and aggregate queries about the statistics of total number of flows by source and total number of unique destinations by source. Finally, simple threshold sampling rules can be used to select a small number of records from sources that have sent more than $T_{total}$ number of flows or contacted more than $T_{unique}$ destinations. This allows for the identification of individual sources that have exceeded the set thresholds, thereby answering the identity queries about the outliers along these metrics.

In this section we have designed a solution that answers the aggregate, distributional, and identity queries for two metrics, the total number of flows by source and the number of unique destinations contacted by each source. The following section discusses how this approach can be further generalized to design a comprehensive system for routine analysis of flow data.

## IV. A COMPREHENSIVE SYSTEM FOR ROUTINE ANALYSIS

The solution designed in the previous section used two arrays of counters and a Bloom Filter for resolving uniqueness

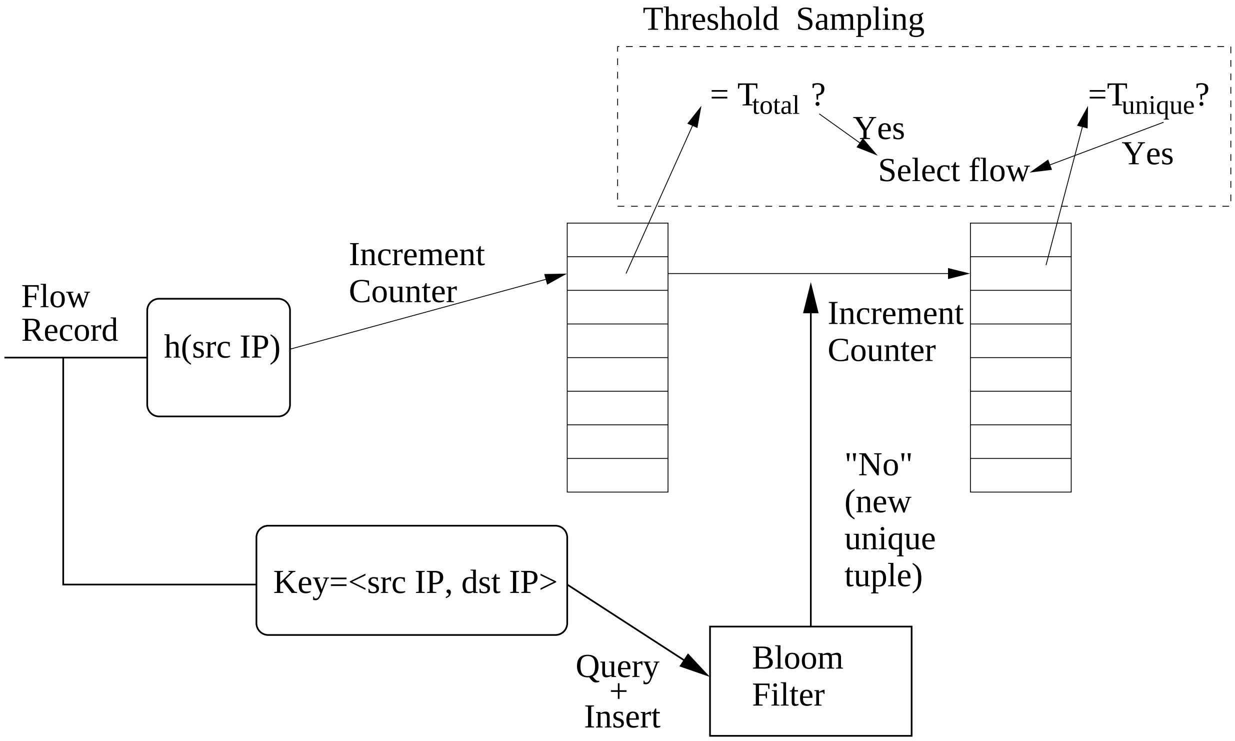


Fig. 1. Solution to track total flows and total unique destinations by source IP

| Metric | Key field(s) | Aggregate Queries | Distributional Queries | Identity Queries |
|---|---|---|---|---|
| Bytes | Flow (5-tuple) | Total Flows | Flows with $x \leq$ bytes $\leq y$ | Large flows |
| Packets | Flow (5-tuple) | Total Flows | Flows with $x$ packets | Large flows |
| Total Flows | Source IP | Total sources | Sources sending $x$ flows | Sources sending many flows ($> T_{total}$) |
| Unique Destinations | Source IP | Total sources | Sources contacting $x$ destinations | Sources contacting many destinations (i.e., contacting $\geq T_{unique}$ destinations) |
| Total Flows | Destination IP | Total destinations | Destinations receiving $x$ flows | Destinations receiving many flows |
| Unique Sources | Destination IP | Total destinations | Destinations contacted by $x$ sources | Destinations contacted by many sources |
| Total Flows | DDP-tuple | Total DDP-tuples | DDP-tuples receiving $x$ flows | DDP-tuples receiving many flows |
| Unique Sources | DDP-tuple | Total DDP tuples | DDP-tuples contacted by $x$ sources | DDP-tuples contacted by many sources |

TABLE II

METRICS TRACKED BY PROPOSED SYSTEM AND EXAMPLES OF SUPPORTED QUERIES.

of tuples to answer queries along two important metrics: the total number of flows and unique destinations by source IP. This solution can be generalized to comprehensively support routine analysis queries. Two observations facilitate this generalization: (i) The total number of metrics that are interesting enough to be tracked on a routine basis are very small, and (ii) A single Bloom Filter can be used universally to resolve all uniqueness questions.

Given the high dimensionality of flow data, it is somewhat counterintuitive that only a few metrics are of routine interest. Indeed, in a forensic investigation, an analyst might need to slice up the flow data along unusual dimensions. However, the key point here is that for *routine* analysis, only a small number of metrics are actually tracked. In practice, information about the eight metrics listed in Table II is adequate to provide situational awareness with enough sensitivity to flag most anomalies. This table also provides examples of queries supported by the sketches tracking the respective metrics. The abbreviation DDP-tuple stand for a `<destination IP, destination port, protocol>` tuple.

One may argue with the exact composition of the list of metrics in Table II, and we do believe that it can be improved with inputs from the community of analysts, but the interesting point here is that all the metrics in this list can be covered by eight corresponding arrays (sketches). The first two arrays, would track the number of packets and bytes on a per flow basis, in an approximate manner. To accommodate a huge number of flows in an array with fewer counters than flows, we can resort to multi-resolution techniques as presented in [4], using 1 million counters to track up to 50 million flows. The remaining six metrics can be tracked comfortably by arrays with $2^{18}$ or 256k counters each. With 32 bit counters, the overall size of these data structures would be 14 MB.

Threshold sampling would identify a small number of records that are interesting because they correspond to an outlier for one of the metrics. Note that for routine analysis, the goal is to identify a small number of interesting outliers. We assume that the detailed data will be available independently for subsequent forensic analysis. For 50 million flows in a measurement, we conservatively estimate $2^{16}$ or 64K flow records, easily fitting within 2MB before compression, using a compact representation such as the SiLK raw format.

The second observation is that a single Bloom Filter can be used to determine uniqueness of tuples across the entire system. Bloom filters use an array of bits, or bit-vector, and a collection of $k$ hash functions to insert and query for elements. For insertion, all $k$ hash functions are computed and the corresponding bits are set to '1'. Upon being queried for a key, again all $k$ hash functions are computed over the key, and the corresponding bits looked up. If all the bits are '1', the answer is "yes", i.e. the key was inserted previously, while if one or more bit is '0', the answer is "No", implying the key is a new unique item. Now, if the hash functions are chosen so they can take variable length inputs, the rest of the operations are transparent to the semantics of the keys, hence allowing us to use the same Bloom Filter for determining the uniqueness of various tuples with different fields. Continuing with the goal of processing 50 million flow records in one period, a Bloom Filter with a single hash function and 16 MB of storage would be sufficient to determine the uniqueness of various tuples with acceptable accuracy. If additional memory is available, these parameters can be tuned to provide optimal accuracy according to the method provided in [7]. Note that the Bloom filter is used as a local data structure used only while updating the sketches for each flow record. Once this process is complete, the Bloom Filter can be discarded; it does not need to be stored or transmitted.

The sketches and selected records corresponding to outliers together make up the summary representation of the flow records. Here, they correspond to a total size of 16 MB before compression. For our example of 50 million flows in an observation period, this is roughly 100 times smaller than the actual flow records and can be transmitted easily to a central analysis server. The analysis server can then compute various estimates over these sketches, feeding a situational awareness or anomaly detection system. The next section discusses the benefits and limitations of this solution approach.

## V. Benefits and limitations

As identified before, the solution approach presented here is complementary to the more flexible flow analysis paradigm and tools available to analysts today. The benefits of this approach lie in its speed and succinctness, while its limitations lie in the restricted set of queries that such a system can support.

- **Speed:** The first major benefit of the proposed approach is its high speed. Updating the sketches involves a small number of hash computations and memory lookups. A software implementation running on a 2.0 Ghz dual-Opteron system can process 2.5 million records per second, preparing the 16 MB of summaries to be shipped to the central analysis server. Running estimation algorithms at the analysis server is equally fast, taking under one processor-minute per sketch.
- **Succinctness:** Large flow monitoring installations typically have a distributed deployment, with *collectors* deployed at various locations in a network collecting flow records from adjacent router(s) and packing them into more succinct intermediate formats, before transmission to a central analysis installation. The approach presented in this paper enables creation of succinct summaries at distributed collectors that are about 100 times more compact than the packed formats of SiLK. This changes the issue of transmission bandwidth (and storage at the central server) from a major concern to a trivial overhead. Indeed in current installations, the large volume of flow records being transmitted, especially during attacks, are a major overhead in large ISPs, and cited as one of the main reasons against the deployment of flow-collection in the core.
- **Low Flexibility:** The benefits of this approach come with a significant limitation - the analysis only addresses a set of predefined queries. This implies that any forensic investigation into network events is likely to require evaluation of queries not supported by the sketches generated into this system. But this problem can be addressed by retaining the complete flow records at the distributed "collectors" till such time when a forensic investigation requires records from the corresponding period to be "pulled" to the central analysis station. Since such investigations are likely to be infrequent, at least relative to the repetitive evaluation of routine queries, such a 2-tier solution will provide all efficiency benefits of the sketch-based system while making available to the analyst all the power and flexibility of conventional flow analysis tools during specific investigations.

## VI. Conclusions

Routine flow analysis tasks that periodically evaluate a fixed set of queries over the flow data collected in the corresponding period can be made significantly more efficient if approximate answers are acceptable in lieu of deterministic accuracy. The approach presented in this paper delivers a 100 fold reduction in the amount of data sent to a central analysis server for routine analysis. Future work on this subject includes the identification of important metrics to track and performance study of a complete, deployed system.

## References

[1] D. Kompanek and M. Thomas, "Silk analysis suite," http://sourceforge.net/projects/silktools, 2003.

[2] C. Gates, M. Collins, M. Duggan, A. Kompanek, and M. Thomas, "More netflow tools: For performance and security," in *Proc. of LISA XVIII*, Atlanta, GA, Nov. 2004.

[3] J. McHugh, "Sets, bags and rock and roll," in *Proc. of the Ninth European Symposium on Research in Computer Security*, Sept. 2004.

[4] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *Proc. ACM SIGMETRICS*, June 2004.

[5] A. Kumar and J. Xu, "Sketch guided sampling – using on-line estimates of flow size for adaptive data collection," in *Proc. of IEEE Infocom*, Apr. 2006.

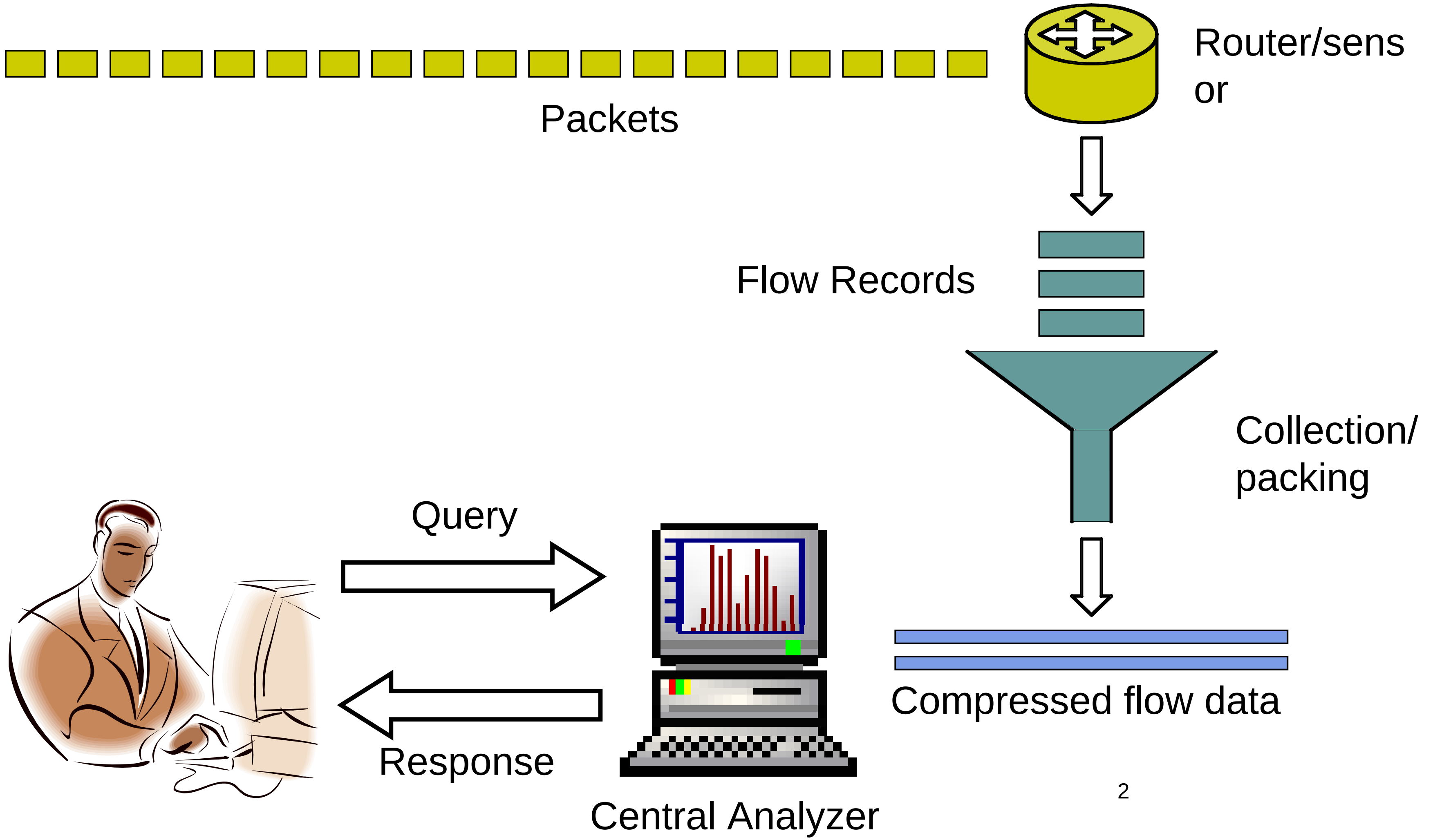
[6] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *CACM*, vol. 13, no. 7, pp. 422–426, 1970.

[7] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," in *Fortieth Annual Allerton Conference on Communication, Control, and Computing*, 2002.

# Scalable Flow Analysis

Abhishek Kumar
Sapan Bhatia
Georgia Tech

# Flow Collection and Analysis Architecture



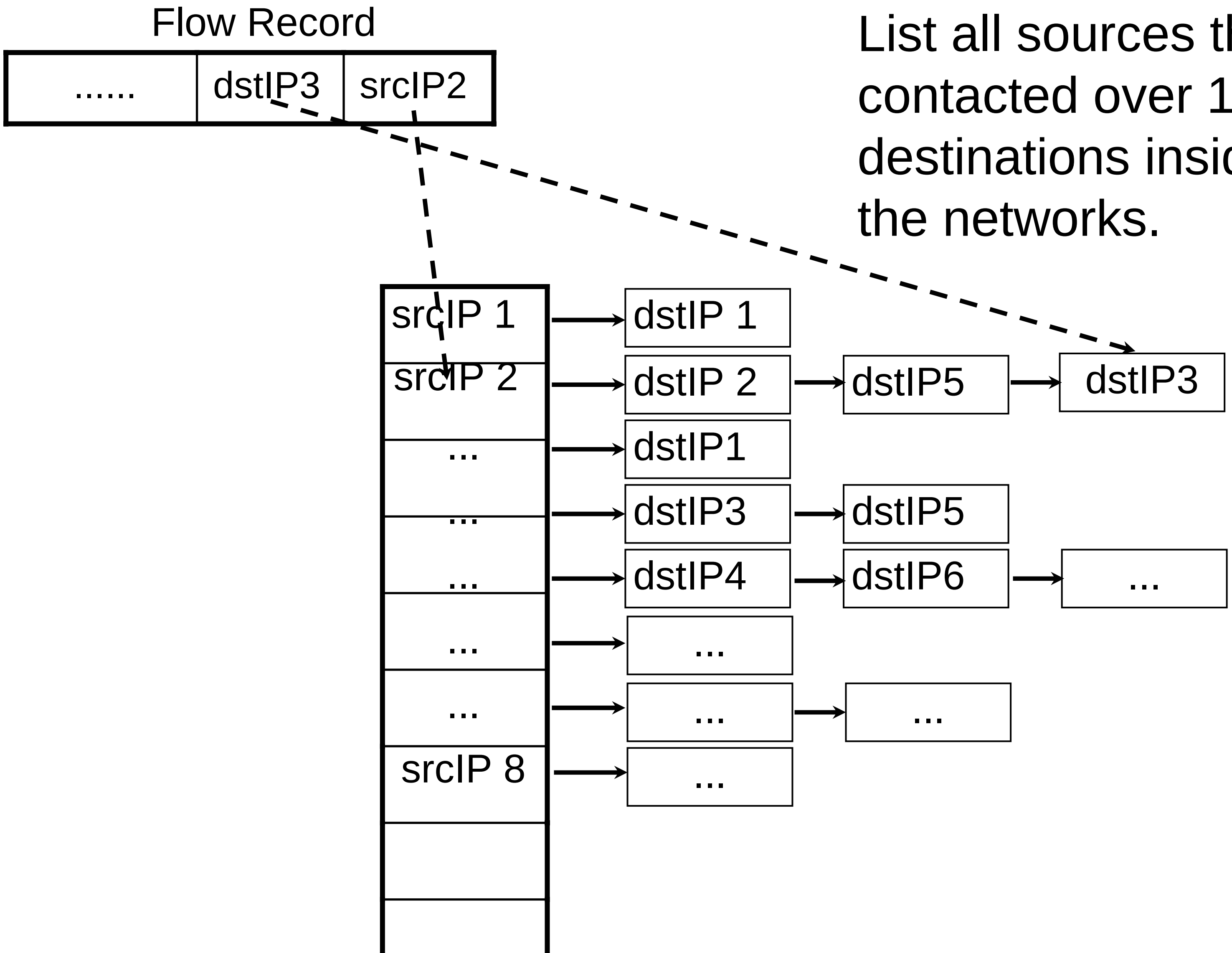
# An informal Taxonomy

[illegible]

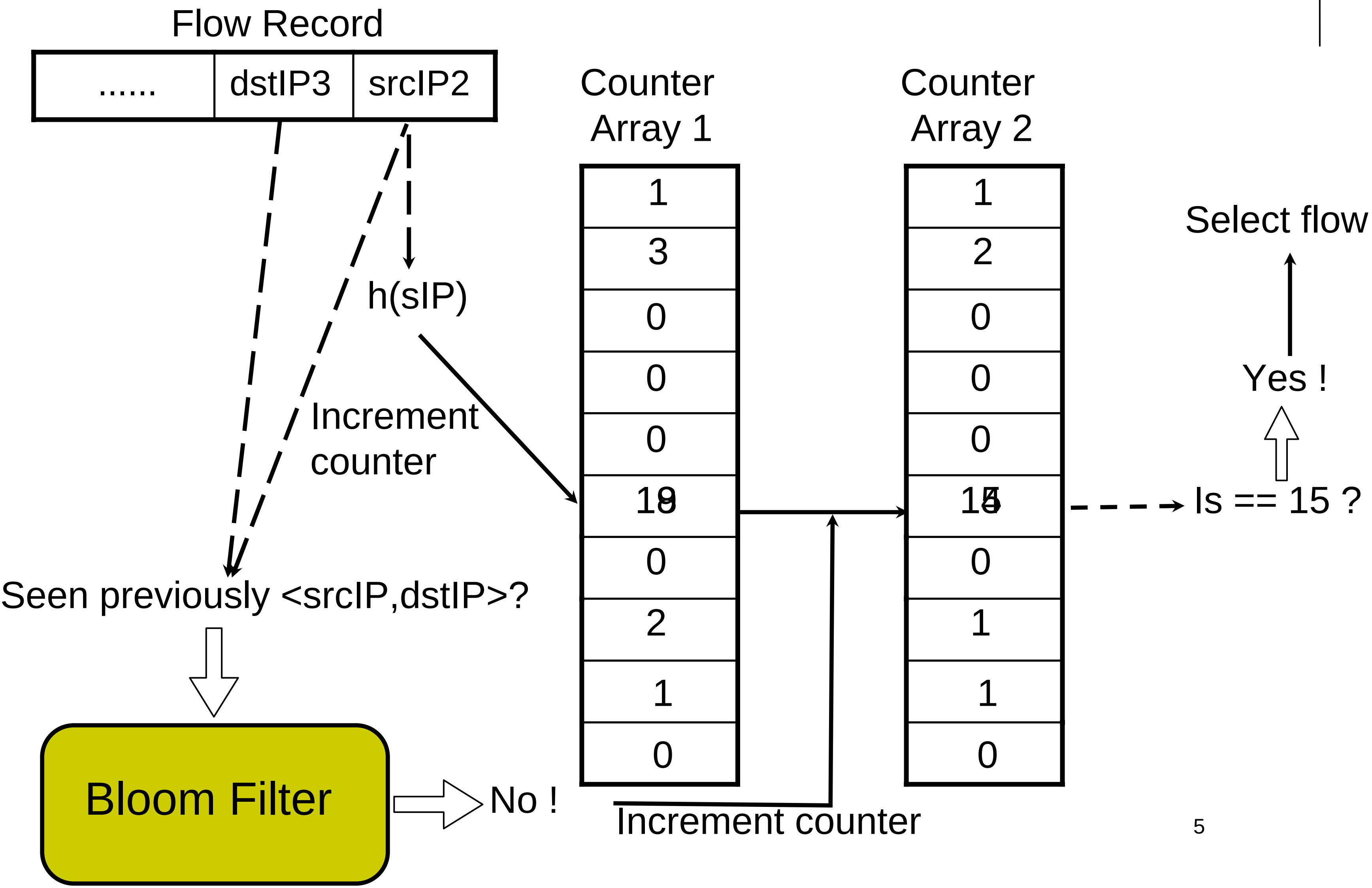
|  | Aggregate Query | Distributional Query | Identity Query |
|---|---|---|---|
| Routine Query |  |  |  |
| Drill-down (forensic) Query |  |  |  |

# An example query

List all sources that contacted over 15 destinations inside the networks.



# Proposed Solution (Preprocessing)



Flow Record

| | | |
|---|---|---|
| …… | dstIP3 | srcIP2 |

h(sIP)

Increment counter

| Counter Array 1 | Counter Array 2 |
|---|---|
| 1 | 1 |
| 3 | 2 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 19 | 15 |
| 0 | 0 |
| 2 | 1 |
| 1 | 1 |
| 0 | 0 |

Increment counter

Is == 15 ?

Yes !

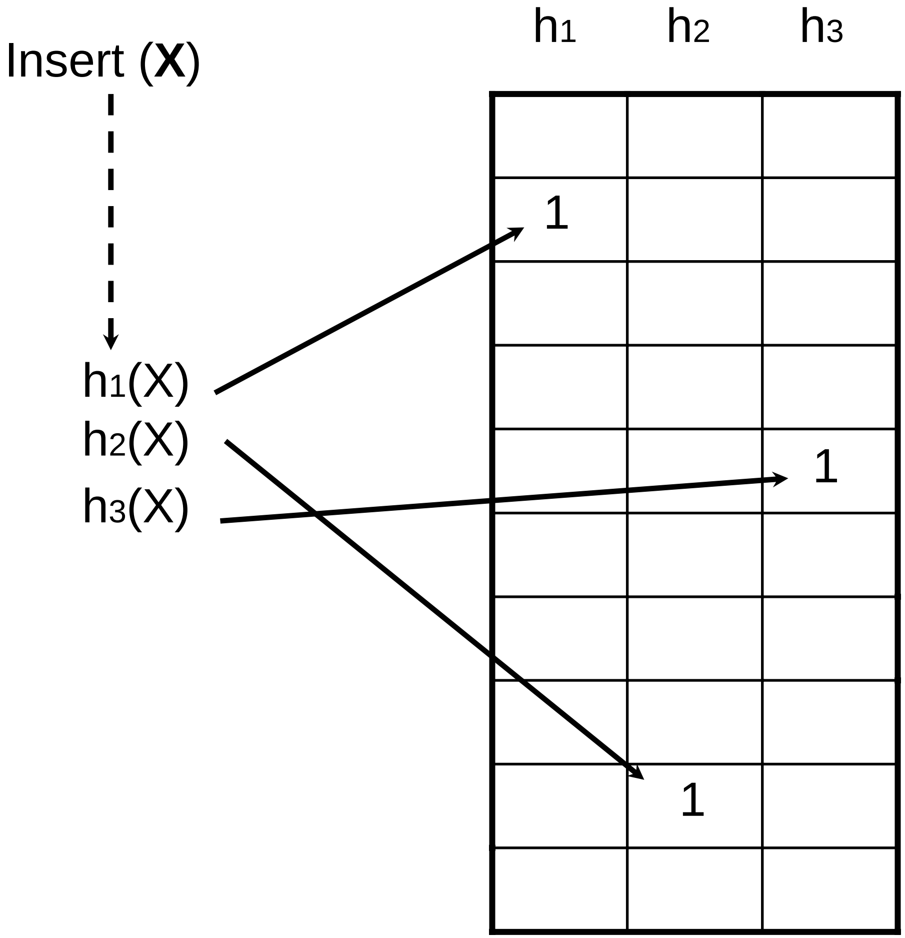
Select flow

Seen previously <srcIP,dstIP>?

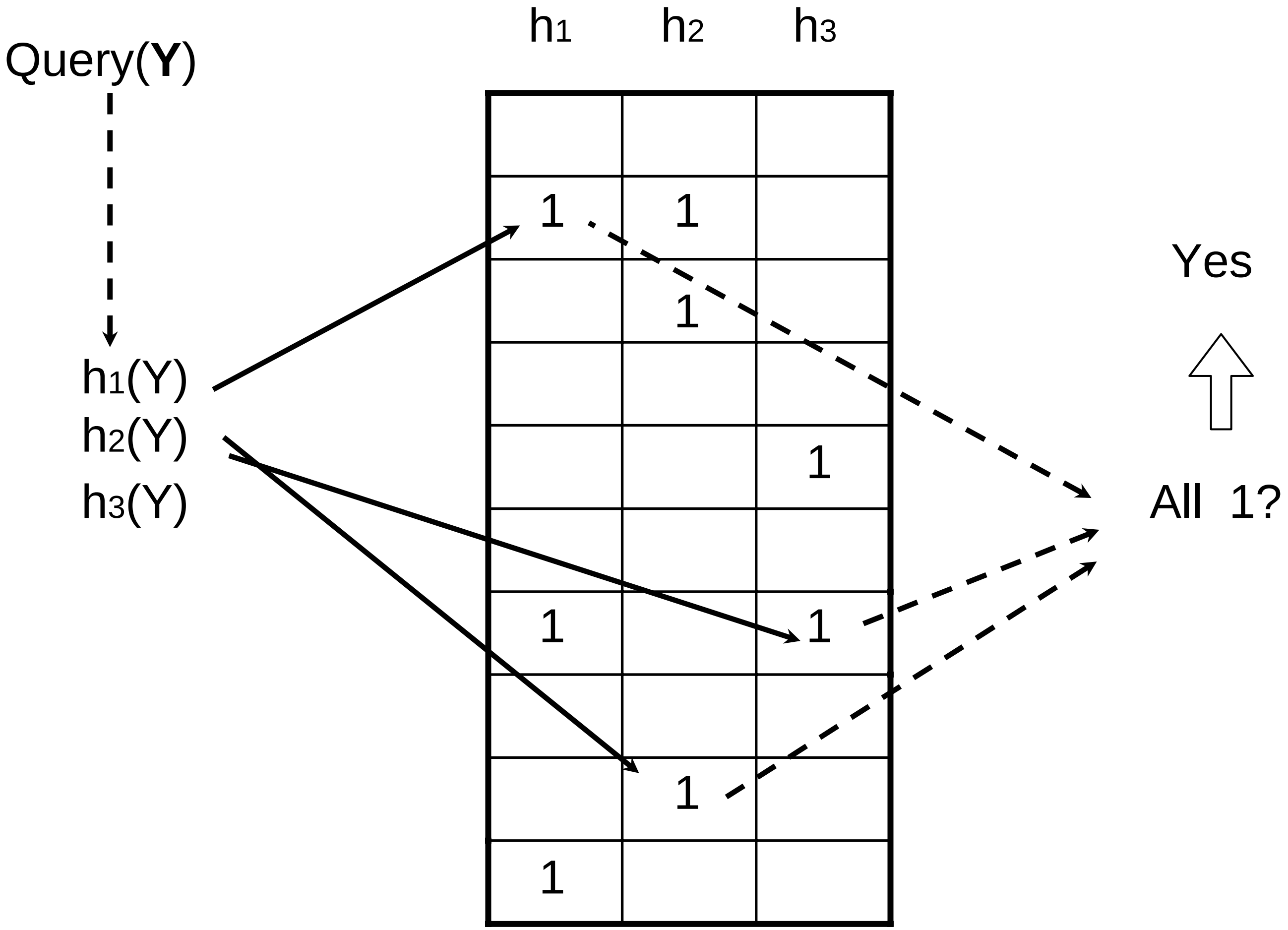
Bloom Filter

No !

# Bloom filter (insert)



# Bloom filter (query)



Query(**Y**)

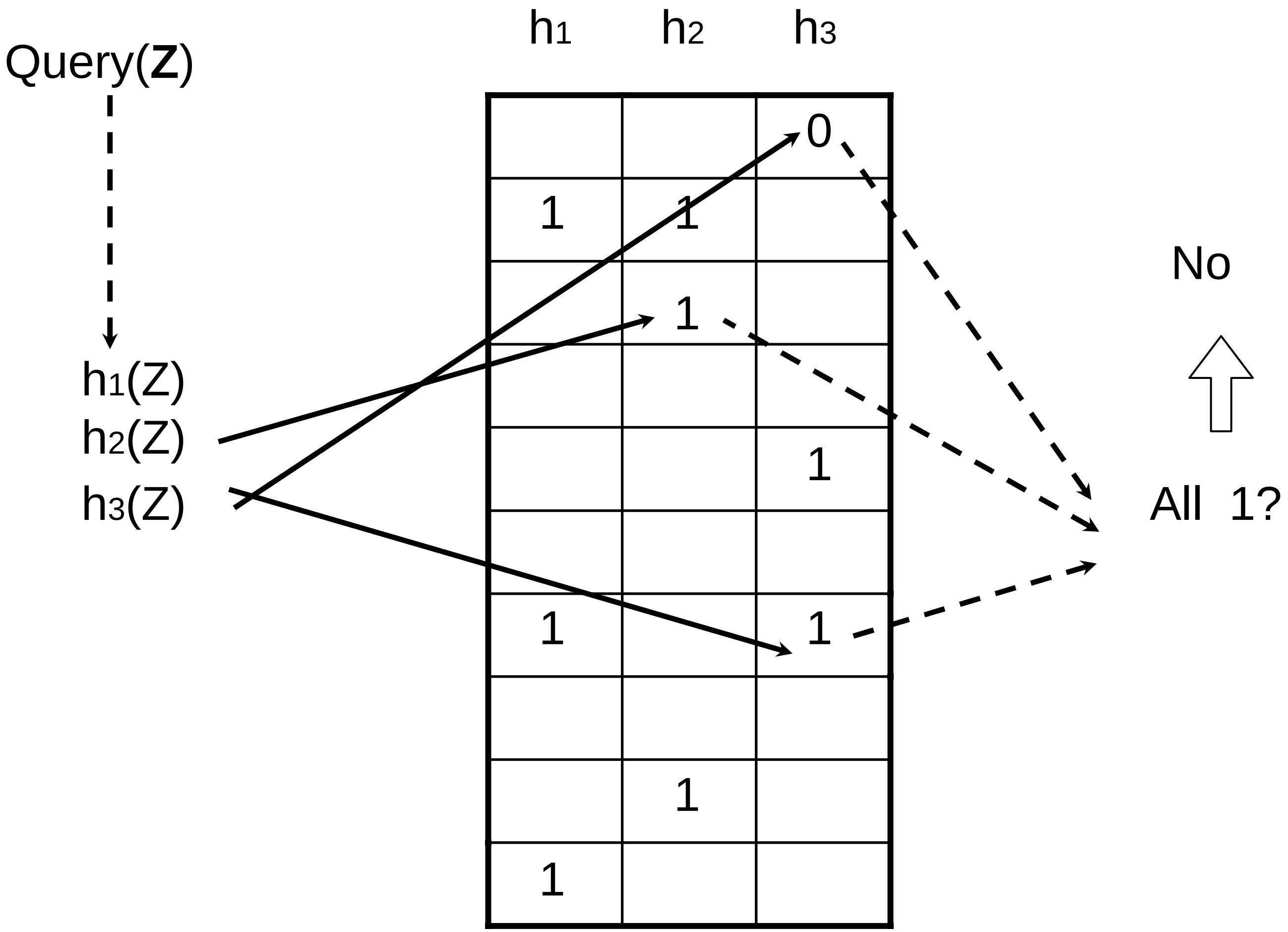
$h_1$(Y)
$h_2$(Y)
$h_3$(Y)

| $h_1$ | $h_2$ | $h_3$ |
|---|---|---|
| | | |
| 1 | 1 | |
| | 1 | |
| | | |
| | | 1 |
| | | |
| 1 | | 1 |
| | | |
| | 1 | |
| 1 | | |

Yes

All 1?

# Bloom filter (query)



Query(**Z**)

$h_1(Z)$
$h_2(Z)$
$h_3(Z)$

| $h_1$ | $h_2$ | $h_3$ |
|---|---|---|
|  |  | 0 |
| 1 | 1 |  |
|  | 1 |  |
|  |  |  |
|  |  | 1 |
|  |  |  |
| 1 |  | 1 |
|  |  |  |
|  | 1 |  |
| 1 |  |  |

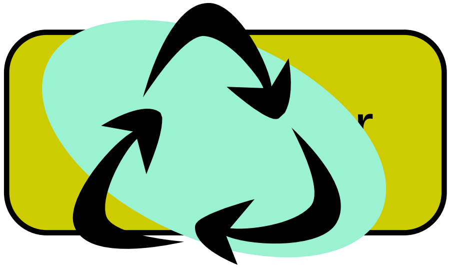
All 1?

No

# Proposed Solution (State after preprocessing)

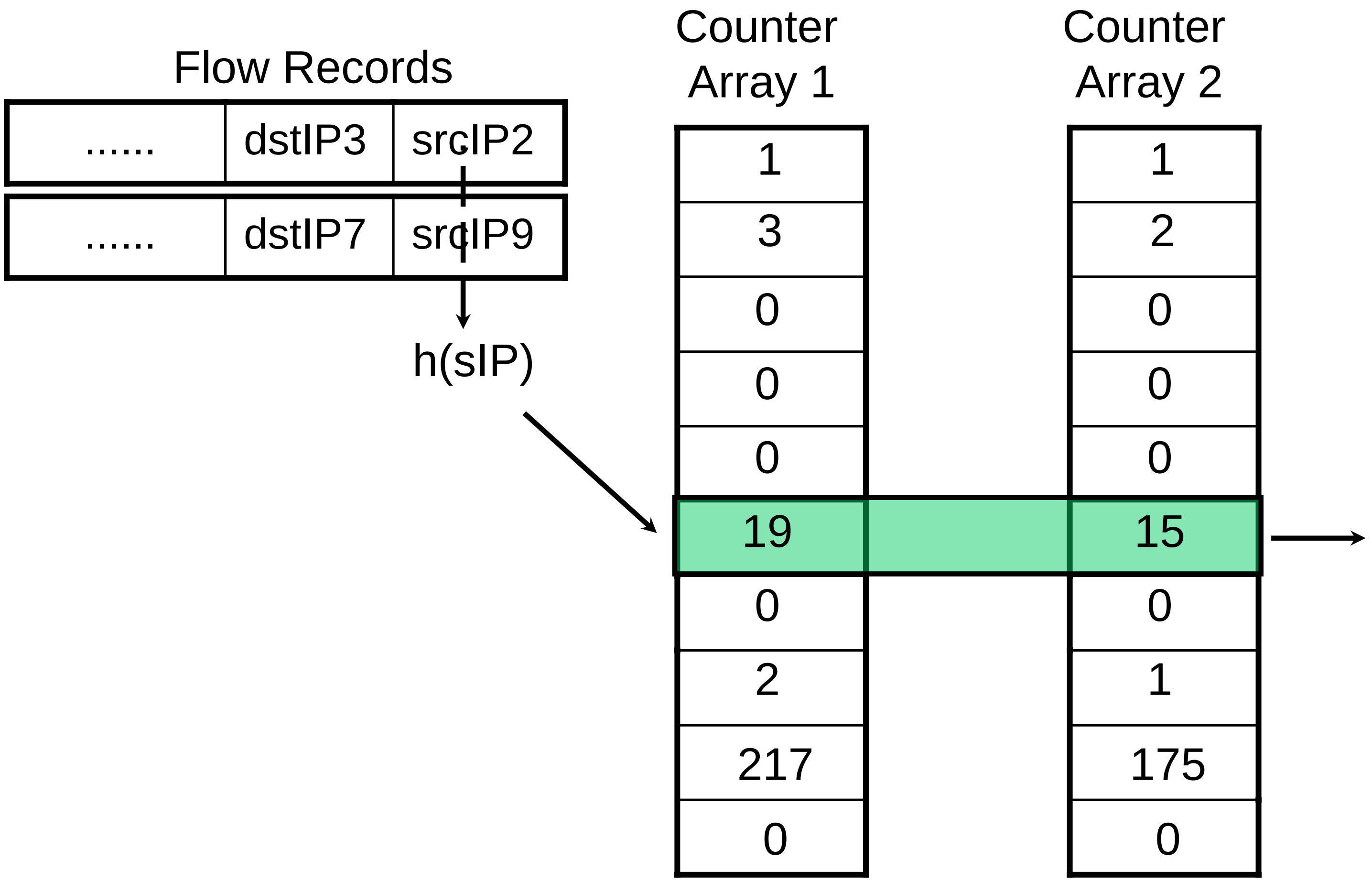
Flow Records

| | | |
|---|---|---|
| ...... | dstIP3 | srcIP2 |
| ...... | dstIP7 | srcIP9 |



| Counter Array 1 | Counter Array 2 |
|---|---|
| 1 | 1 |
| 3 | 2 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 19 | 15 |
| 0 | 0 |
| 2 | 1 |
| 217 | 175 |
| 0 | 0 |

# Proposed Solution (Query processing)



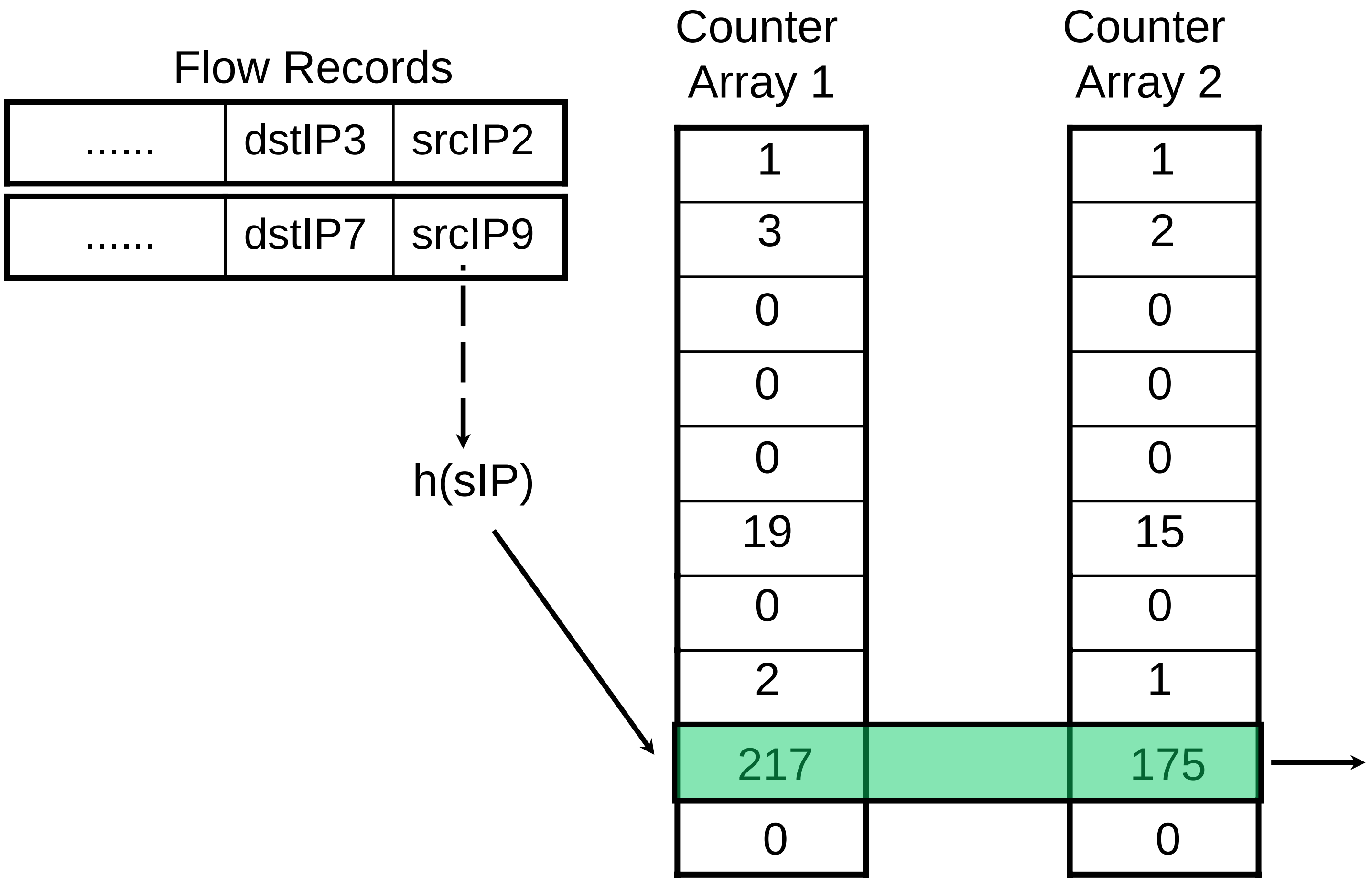
Flow Records

| | | |
|---|---|---|
| …… | dstIP3 | srcIP2 |
| …… | dstIP7 | srcIP9 |

| Counter Array 1 | Counter Array 2 |
|---|---|
| 1 | 1 |
| 3 | 2 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 19 | 15 |
| 0 | 0 |
| 2 | 1 |
| 217 | 175 |
| 0 | 0 |

Source sIP2:
Total flows ~ 19
Unique dsts ~ 15

# Proposed Solution (Query processing)



Flow Records

|  |  |  |
|---|---|---|
| …… | dstIP3 | srcIP2 |
| …… | dstIP7 | srcIP9 |

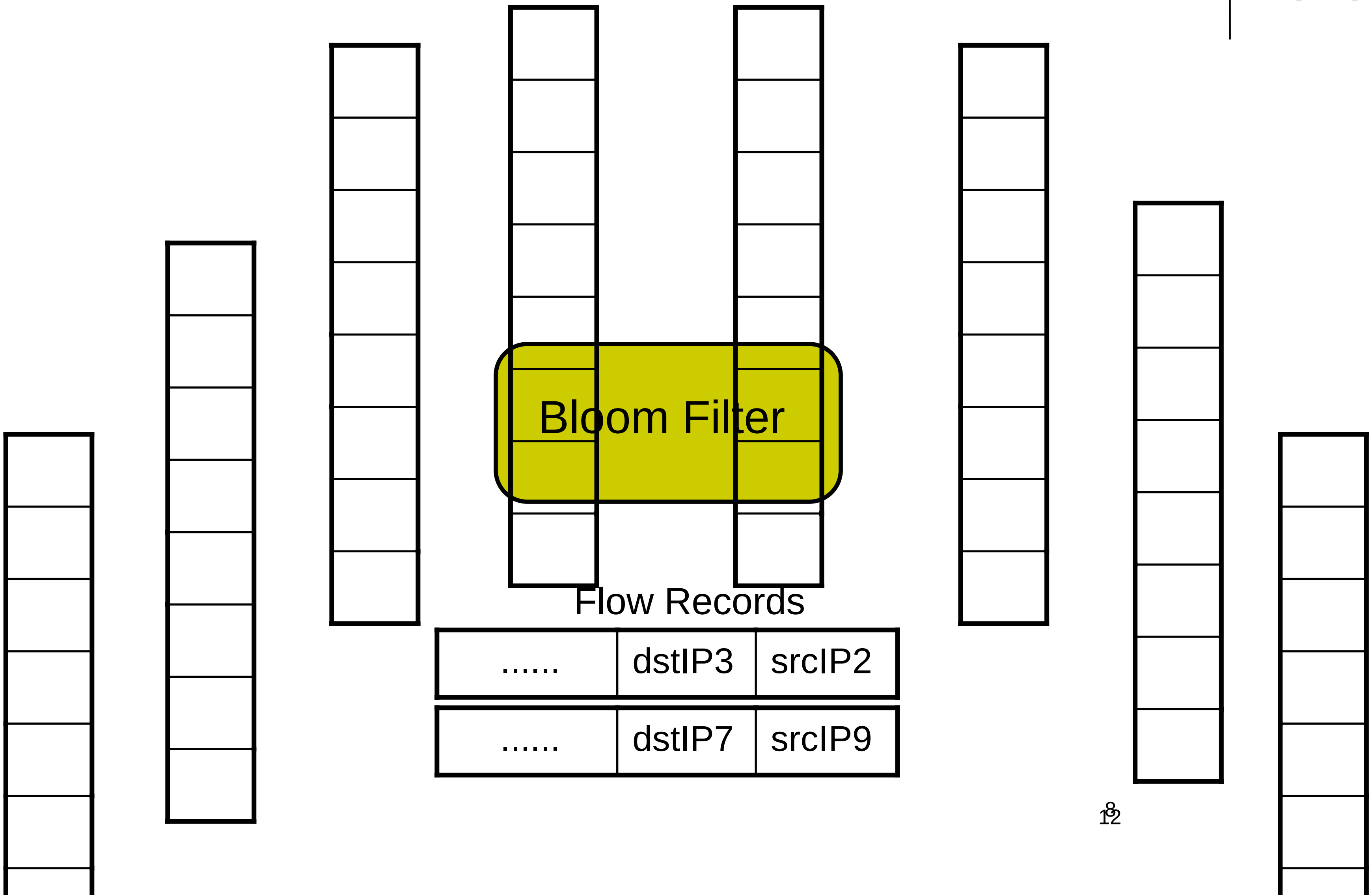
h(sIP)

| Counter Array 1 | Counter Array 2 |
|---|---|
| 1 | 1 |
| 3 | 2 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 19 | 15 |
| 0 | 0 |
| 2 | 1 |
| 217 | 175 |
| 0 | 0 |

Source sIP2:
Total flows ~ 19
Unique dsts ~ 15

Source sIP9:
Total flows ~ 217
Unique dsts ~ 175

# Can we build a more comprehensive system ?



| | | |
|---|---|---|
| …… | dstIP3 | srcIP2 |
| …… | dstIP7 | srcIP9 |

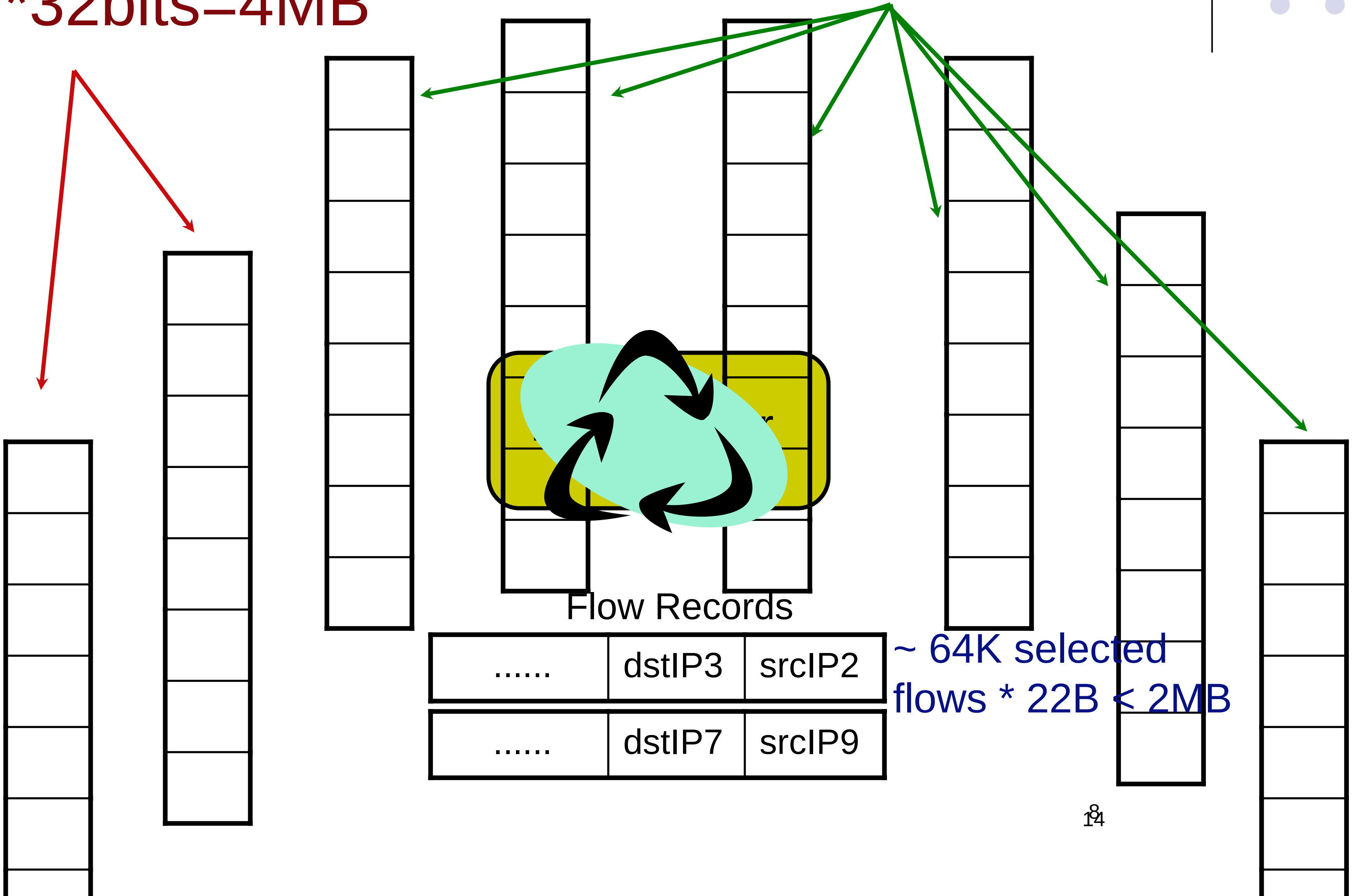
# What will it track ?

| Metric | Key Field(s) | Aggregate Queries | Distributional queries | Idenity Queries |
|---|---|---|---|---|
| Bytes | 5-tuple | Total Bytes, Flows | Flows with x<bytes<y | Large Flows |
| Packets | 5-tuple | Total Pkts, Flows | Flows with x Pkts | Large flows by pkts |
| Total Flows | Source IP | Total sources | Sources sending x flows | Sources sending many flows (> Threshold) |
| Unique Destinations | Source IP | Total sources | Sources contacting x destinations | Sources contacting many destinations |
| Total Flows | Dest IP | Total Destinations | Destinations receiving x flows | Destinations receiving many flows |
| Unique Sources | Dest IP | Total Destinations | Destinations contacted by x sources | Destinations contacted by many sources |
| Total Flows | <dIP, dPort, proto> | Total 3-tuples | 3-tuples receiving x flows | 3-tuples receiving many flows |
| Unique Sources | <dIP, dPort, proto> | Total 3-tuples | 3-tuples contacted by x sources | 3-tuples contacted by many sources |

# How much space for 30 M flows?

1M*32bits=4MB

256K*32bits=1MB

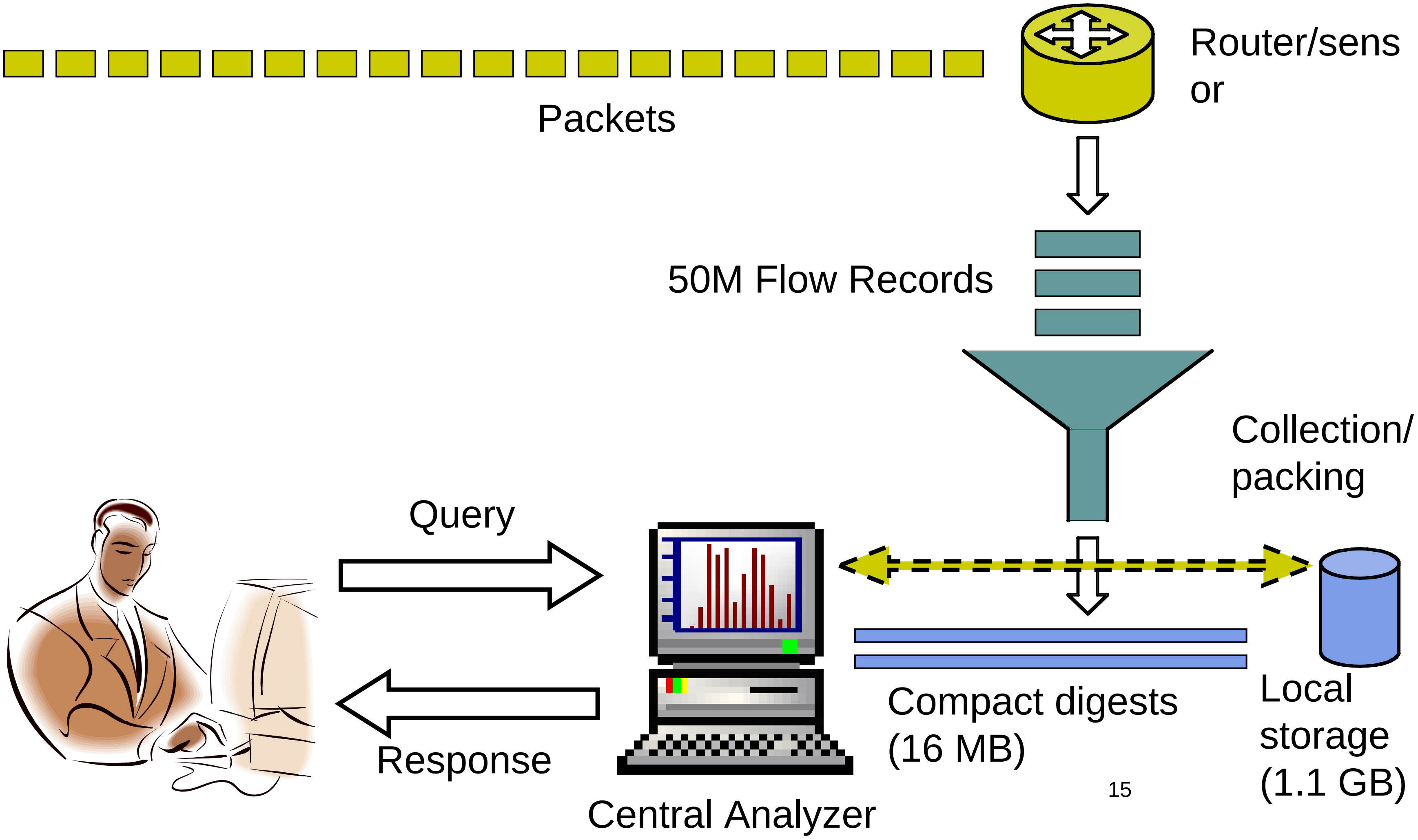


Flow Records

| | | |
|---|---|---|
| ...... | dstIP3 | srcIP2 |
| ...... | dstIP7 | srcIP9 |

~ 64K selected flows * 22B < 2MB

# Flow Collection and Analysis Architecture



# Thank you !

- Questions and comments
- Contact: akumar@cc.gatech.edu

# System Requirements for Flow Processing

**Raj Srinivasan**
**Director, Software Engineering**
**Bivio Networks, Inc.**
**12 Sept 2006**

## Abstract

With the ever growing array of threats to computer systems, security applications have evolved from simple firewalls to extremely complex entities with very sophisticated requirements. These requirements need to address functionality, performance, and scalability. Functionality spans firewalls, VPNs (IPSec and SSL), IDS/IPS, AV/AS, and server off-load functions, among other things. Performance includes both the ability to pass packets through the system at multi-gigabit rates, as well as being able to do deep packet inspection/processing at high speeds. Scalability must address both of these aspects of performance - i.e. packet throughput and processing power. It is also important to consider system management, maintainability, ease of use, and reliability. In this paper, we examine these aspects of the system, and propose an architecture that not only meets these requirements, but will also be flexible enough to support future application needs.

## Introduction

In this paper, we first describe an architecture for processing flows. We then take a look at each aspect of the architecture, and qualitatively describe how it meets the requirements for processing flows in an optimal way.

### Flow Processing Requirements

The following requirements are addressed in this document.

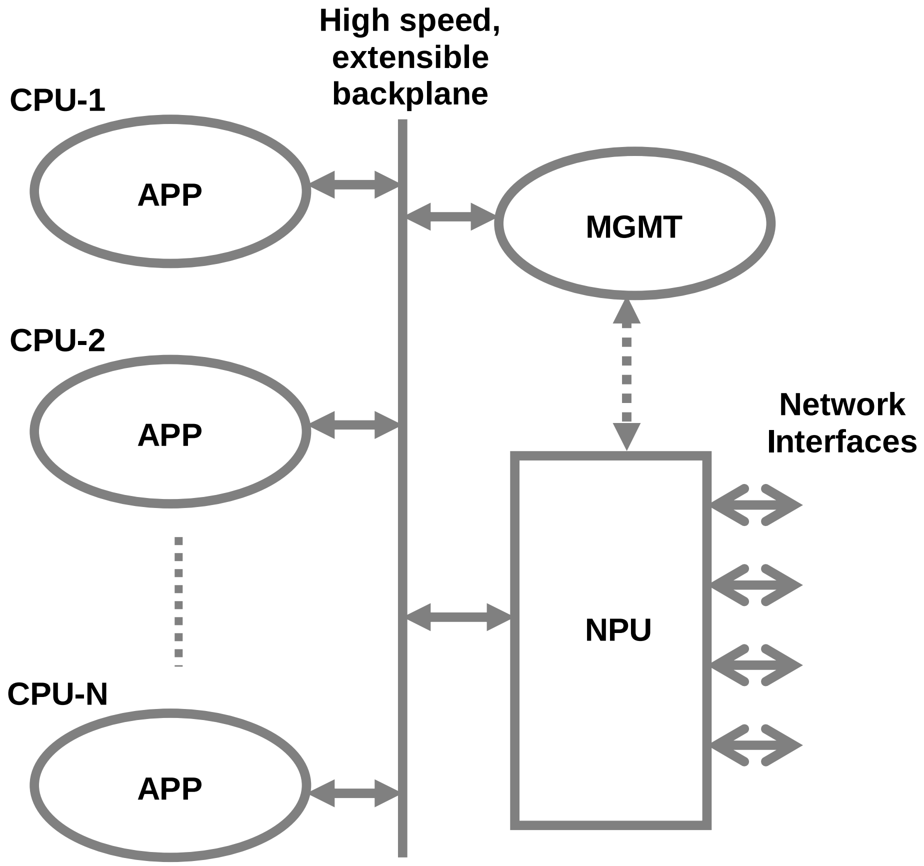
- Performance – while we don’t state a specific number, it is clear that today’s performance requirements are in the multi-gigabit range. Small and large packets must be processed at wire speeds, which could range from 1Gbps to 10Gbps or higher.
- Scalability – the system architecture must be scalable so that different performance requirements can be addressed with meaningful cost structures.
- Functionality – the architecture must be versatile, in that it should support multiple applications. This translates to the requirement to be able to process flows of different types. Flows may constitute single or multiple network associations (i.e. connections, for example) between cooperating applications distributed over a network.

- Manageability – the architecture should be easily manageable. Ease of configuration, management and maintenance are crucial to the adoption of this architecture.
- Application maintenance and portability – when applications are developed for this architecture, they should still maintain a general flavor that will support other architectures. The architecture must shield applications from getting locked into specific hardware or software features.

### System Architecture

The following diagram shows our proposed system architecture.

Architecture for Flow Processing



This architecture utilizes a combination of general purpose processors (CPUs) running the Linux operating system, and a new generation Network Processing Unit (NPU) for performing various flow operations. One processor (MGMT), also running Linux, is reserved for purely management functions, while other CPUs run applications. All of the

CPUs and the NPU reside on, and communicate over, an extensible, high speed backplane that operates at multiples of 10Gbps. The external network interfaces, consisting typically of 1Gbps or 10Gbps Ethernet, are connected to the NPU.

In addition, each application CPU has attached to it one or more co-processors over a very fast bus (PCI Express, for example). These co-processors typically perform such functions as cryptographic encryption and decryption, key generation, regular expression processing, etc.

The MGMT CPU will typically have its own private Ethernet port and console port for management purposes. It will also support storage, for example, in the form of an attached SCSI/RAID disk system.

## The Networking Environment

The system supports one or more network interface cards, with each card supporting multiple ports (for example, eth0, eth1, and so on). All configurations are done on MGMT. Through a process of virtualization, these interfaces are replicated on every application CPU. The system software ensures that interface states are identical on all the processors. Thus, applications have direct access to the external interfaces, from every processor on the system. There is a special Ethernet interface in each CPU – bp0 – that enables applications to communicate over the backplane. The backplane essentially looks like a very high speed, private Ethernet network, not visible from the external network. Applications may communicate with each other over the backplane using standard socket based communication methods. Thus, every processor, including MGMT, has an identical networking environment.

## The File System Environment

Application CPUs all boot using standard protocols from the management CPU, and mount the storage attached to MGMT. Thus all CPUs have identical file system environments.

## NPU Functions

The NPU performs the following functions:

- Load-balance incoming traffic to applications based on multiple algorithms.
- Support bindings, set using special APIs, that enable special flow processing. For example:
  - Cut through flows.
  - Direct specific flows to applications requesting them.
- Perform QoS functions.
- Maintain, and report full flow statistics.

- Support application specific flow processing, either dynamically through APIs, or statically through application specific code, linked to NPU processing through architected hooks for intercepting packets between input and output.

NPU bindings are set by applications over the backplane using protocols designed for this purpose. These bindings must be such that the API calls used to set them must have very low latency.

## Benefits of Proposed Architecture for Meeting Flow Requirements

We are now ready to consider the benefits of this architecture in fulfilling flow requirements. We start by with comparing the architecture with specialized hardware implementations.

### Comparison with Specialized Hardware

One approach to flow processing is to use hardware developed specifically to process flows in certain ways. There is no doubt that specialized hardware will outperform more general systems for specific functions. However, such systems necessitate a number of compromises, among which are:

- Flexibility. Unlike general purpose hardware which is programmable, often ASICs are designed to be used in very limited ways, with limited programmability. These are okay for well established protocols which operate at low levels, but definitely not for generalized flows where applications are constantly evolving. It is not feasible to require a new spin of an ASIC in order to present new features to customers, for most application vendors.
- Scalability. Often, ASICs lack the general purpose hardware’s ability to be clustered, or stacked. Thus scalability needs to be built with external “glue” hardware, often at a huge cost.
- Performance. This may seem like a contradiction, since ASICs are built with performance in mind (in addition to volume considerations for decreasing cost). However, we need to keep in mind that in order to make ASICs support multiple applications, some amount of generality needs to be embedded in it. This makes it more like a CPU, and often, it runs at a fairly low clock frequency. Thus, while a specific function (such as cutting through flows, or bulk encryption) may outstrip the capabilities of generalized hardware, combinations of functions will severely limit performance.
- Application portability. When portions (or all of) applications are changed to use specialized hardware, the application will lose its generality. It is often a huge effort to split an application so that certain portions of processing are delegated to special hardware. This makes it very hard to maintain the applications portability to other systems – for example, in a product range that meets different

performance levels – and locks in the application to the specific hardware components.

Thus, where all of the above requirements are important, and it is of utmost importance to be able to change quickly to conform to new conditions, a more generalized hardware would be the most appropriate way to go. Note that what we say about ASICs applies to FPGAs also.

## Performance

Until recently, simple architectures sufficed for a couple of reasons: firstly, performance requirements have not been very high for flow processing applications, and CPU speeds have been keeping up with Moore's Law. Both of these considerations have undergone major changes recently.

Flow processing speeds are now required to be in the 10Gbps range by service providers. Even though CPU speeds are very high – Intel processors run close to 4GHz – flow throughput is still being limited by factors other than CPU speed. One of the limiting factors is memory access. Even with the fastest memory (DDR2, 600 MHz, 64-bit access), practical considerations limit access to well under 10Gbps. This bandwidth needs to be utilized for processing (running code and accessing local variables), as well as for transferring data in and out of memory. Thus, however fast the CPU may be, memory bandwidth considerations will limit the amount of data that can be processed to a few hundred megabytes of small packets to a few (usually 2) gigabits of large packets. This is true even for SMP clusters, since such a cluster will operate on a single memory (even if distributed among multiple banks using dual controllers).

Moreover, if there is a need to integrate multiple applications – as in the case of UTMs (Unified Threat Management systems) – in order to meet complex flow processing requirements, the amount of processing that needs to be done for each flow increases drastically. Since Moore's law seems to have hit a limit (witness Intel's emphasis on dual cores now) the only way to scale is to use more processors in a loosely coupled scheme. Each processor may be a multi-core unit in the above architecture, so one gets the best of both worlds.

## Scalability

The most common design for a flow processing consists of a CPU subsystem (an Intel dual-core, typically) attached over a fast bus such as PCI Express to an NPU that incorporates a crypto engine, and a regular expression processing engine. Aside from the single system performance considerations we have noted above, one huge drawback of such system is their scalability. The missing piece is really the extensible backplane incorporated in the architecture we have presented. This backplane allows additional CPUs, NPUs, and their associated resources, to be added to the system.

One of the tremendous advantages of the architecture we have presented is that scaling is almost linear when more CPUs are added. This depends on the NPU to perform a clean partitioning of flows, so that the need to synchronize with other CPUs is minimized. We will discuss this some more later.

Also, having specialized engines for cryptographic functions and regular expression processing be attached to each CPU via a fast bus such as PCI Express allows scaling of applications that use these functions. If these resources were combined with the NPU – as do many architectures – two problems present themselves. One problem is that communication between the CPUs and the NPU could become considerable, thus affecting scaling. Secondly, since a given NPU has fixed co-processing capability, this resource cannot be easily scaled.

A good side-effect of our architecture is the built-in redundancy. With multiple CPUs available, if one CPU fails, it can be backed up using multiple schemes. Typically, the NPU will detect this condition, and load-share the flows targeted at the failed CPU among the active CPUs.

## Functionality

By keeping the primary functionality in the application CPUs, and using a generally accepted operating system such as Linux, one maximizes the ease with which new functionality can be introduced. Indeed, there is a plethora of software available in the realm of open and free source for Linux, and it will be fairly easy (usually requiring a compilation only) to port this software to our architecture. Indeed, we have done precisely this in our implementation. Moreover, there are architected means to employ multiple applications in the same CPU, as well as divide CPUs into groups where each group processes flows requiring a specific application.

We are not saying that it is impossible to provide new functionality in other architectures. Only that it will be a non-trivial port, often requiring NPU specialists.

## Manageability

This is very often an overlooked function, which requires considerable resources from the system. In most systems, management has to coexist with applications in the same CPU. If the system is busy, response time will be very poor when an administrator tries to perform configuration or maintenance. This is not acceptable when there are situations which require immediate action (such as a breach of security).

In our architecture, this problem is solved by dedicating a CPU (MGMT) which is solely used for management purposes. The CLI and GUI run on this CPU. It has its own, dedicated, management Ethernet port over which the system is always accessible and response is guaranteed to be immediate. This CPU can be used to consolidate and preprocess log information and statistics, if necessary, thus freeing the application CPUs for flow processing.

One very important function performed by the management CPU is general monitoring of system health. It watches hardware, the system environment (such as temperature), and the running applications. When a failure is detected, the criticality of the failure will be assessed, and corrective action will be taken immediately. This action may be to restart a failed application, shutdown a non-functioning CPU, signal a secondary (active or standby) system to take over, reroute traffic on a failed link, or failopen interfaces (for an IPS application, for example) to take the system offline due to non-recoverable errors.

Thus, the importance of a clean separation of management functions from applications cannot be overemphasized.

## Application Maintenance and Portability

Common sense dictates that this is an important consideration for commercial ventures. The architecture we have proposed ensures this by providing a full Linux environment for applications, requiring only minimal changes, if any, for functioning in this environment. Indeed, we have ported open source applications to this architecture with only a recompilation. A beneficial side-effect is that applications do not get locked in to a particular architecture.

## Some Additional Scaling Considerations

We said before that scaling using this architecture is linear. In this section, we expand on this statement.

Scaling is a term that applies to multiple features of a system. It can apply, for example, to network throughput, or to compute power. This architecture promotes scaling in both dimensions.

In order to increase the network throughput of the base system, an additional NPU can be added to the extended backplane. In practice, the additional NPU would be part of a board that might contain additional CPUs. The effect of the additional NPU is to increase the IO capability of the system. We have done this on a similar architecture with very little change to the base code. The main requirement is that the system be aware of the NPUs and their addresses on the backplane.

To increase the available compute power, additional CPUs can be added to an extended backplane. This enables more processing for each flow, for a given maximum system throughput. With enough compute power, wire-speed can be attained for any type of flow processing within reason (it is always possible to create pathological examples that are hard to accommodate).

In an SMP configuration of CPUs, we already saw how memory bandwidth affects throughput. Inherently, scaling using SMP configuration is not linear, because of the effects memory contention, and use of locking to arbitrate contention for critical resources.

In the loosely coupled CPU architecture we have described, scaling can be linear if the NPU is used appropriately. Essentially, by recognizing and grouping associations into sets of related flows that are processed by the same CPU, the need for synchronization between CPUs is drastically minimized. Synchronization may still need to be done for slow-path control functions such as configuration, but these do not affect the fast-path data processing. Aggregation functions such as statistics over the full set of flows (not visible to a particular CPU in its entirety), detecting certain types of anomalies, QoS, etc. are good candidates for implementation in the NPU. Not requiring CPUs to synchronize for these features makes CPU scaling linear.

## Summary

In this paper, we have described an architecture which we believe is most suitable for general flow processing.  Requirements for flow processing were enumerated, and an architecture for implementing these requirements was proposed. Each aspect of the architecture was discussed qualitatively, and shown to be of benefit in realizing flow processing requirements.

We have implemented this architecture and it has been deployed for processing flows commercially.

## A Traffic Analysis of a Small Private Network Compromised by an On-line Gaming Host

Ron McLeod, BCSc, MCSc.
Director - Corporate Development Telecom Applications Research Alliance
Doctoral Student, Faculty of Computer Science, Dalhousie University

## Abstract

In the early months of 2006 a small private network (the Network) suffered a noticeable degrading of its network performance. A network traffic capture and analysis was conducted and used to investigate the network performance issues. This paper presents partial results of that analysis. The network traffic capture formed part of an experimental use of the SilkTools$^{tm}$ [1] capture and analysis suite developed by CERT personnel at Carnegie Mellon University. During the first analysis of the captured data it was discovered that the Network contained a host that had been compromised at some time in the past and was currently being used to support the on-line gaming activity of over 174,000 distinct player source addresses around the globe. These players were believed to be participating in the Half-life$^{tm}$ [2] first-person shooter game (the Game). The initial finding was the result of a manual investigation of unusual time and volume traffic spikes from arbitrarily chosen time slices. Subsequent work was conducted on searching for a traffic signature which could be representative of the presence of the Game such that future discovery of Game activity could be automated. Gaming traffic is predominantly UDP traffic of high byte volumes, typically targeted at a given range of destination ports. This analysis also searches for a specific TCP traffic pattern that is suggestive of a Game signature. Network traffic patterns that emerge after access to the compromised host has been closed are labeled as SCAR traffic, for **S**evered **C**onnection **A**nomalous **R**ecords

### 1. Methodology and Experimentation

### 1.1. The Network Sampling Environment

On February 3, 2006 an ongoing traffic capture was initiated within the Network. This was accomplished by instructing the primary edge router to construct netflow [3] records and to deliver those records to a single collection point within the Network. The Network used for experimentation is comprised of four /24's, one of which has been divided into /27's. In total, the entire Network consists of approximately 40 user assigned hosts, although the actual number of hosts is subject to minor fluctuations over time. Approximately an additional 40 special purpose hosts also exist within the address space although these hosts are not assigned to individual users. Many of the hosts are the property of separate owners and are subject to separate administration. However, traffic from each passes through a single edge router and it is at this router that the author did his data collection. For reasons of privacy, payload data was neither collected nor examined. It was further understood that the author would not have access to the content of specific hosts for further investigation purposes. Although, the owners of each host for which anomalous activity (if any) was discovered would be informed immediately of any observed anomaly in their machines and full disclosure of the analysis results would be made upon request. For confidentiality reasons the identity of the Network is not specified in this document. For purpose of analysis, only the non-port 80 traffic and non-null traffic was initially considered.

### 1.2 Network Analysis – The Discovery of an Intruder

On February 11, 2006 the first sample of network traffic was extracted for analysis. The time period from midnight to 7:00 AM local time on February 8 was chosen for the first data slice. This was partially a random choice and partially due to the fact that the author expected minimal traffic volumes during this time. The analysis tools were instructed to access each flow record for the time in question and to extract Source IP address, Destination IP address, Source Port, Destination Port, Protocol, Bytes (the number of bytes in the flow record), TCP Flags, Start Time (of the flow record) and End Time (of the flow record).

Figure 1 shows a profile of the sampled traffic data that corresponds to a three tuple consisting of the Flow Record Number (in order of appearance), Protocol Number and Bytes per flow record for the hour of 12:00 – 1:00 AM.

Figure 1

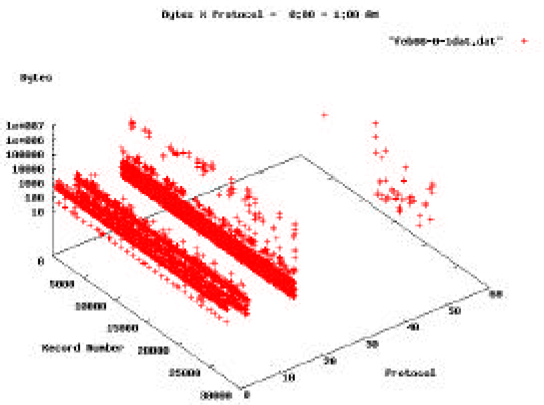


Figure 1 shows a clear grouping of the data into four distinct bands corresponding to five separate protocol clusters. They are:

1. Protocols 50 and 53 used by IPSEC and SWIPE respectively.
2. Protocol 17 used by UDP traffic.
3. Protocol 6 used by TCP.
4. Protocol 1 used by ICMP.

Table 1

| Port | Flows |
|---|---|
| 53 | 260596 |
| 123 | 16139 |
| 137 | 37586 |
| 138 | 26875 |
| 161 | 40799 |
| 500 | 28151 |
| 1027 | 10170 |
| 1031 | 18241 |
| 1954 | 13445 |
| 2008 | 11777 |
| 2967 | 51571 |
| 5060 | 81821 |
| 6346 | 16320 |
| 25383 | 141890 |
| 26900 | 72348 |
| 27000 | 13173 |
| 27001 | 13342 |
| 27002 | 13174 |
| 27003 | 13233 |
| 27005 | 34933 |
| 27010 | 13039 |
| 27015 | **6061263** |
| 27243 | 64616 |
| | |

Within these bands the largest consistent byte volume is within the UDP (Protocol 17) Band.

The first point of interest was the volume of flow records. There were approximately 27,000 records between 12 midnight and 1:00 AM, when no users were present. The records were then ordered by byte size. This was the first attempt to search for outliers within the data.

From this sorting it was discovered that a small group of SourceIPs using protocol 17 appeared to be responsible for a large portion of the traffic bytes. These were referred to as the Heavy-Traffic-Hosts. However, given the size of the database it was not immediately apparent if there was a subset of the Heavy-Traffic -Hosts that were unusually heavier than the rest. The traffic was then sorted by Source IP and the total bytes over all flow records were accumulated for each SourceIP. The result was striking. One SourceIP accounted for more than 56% (79,865,126 bytes) of the traffic volume measured in bytes during the hour in question. This SourceIP was labeled as the Suspicious Host

The next step in the analysis was to examine the remaining available data about Suspicious Host. It was found that out of 27,477 flow records for the hour, the Suspicious Host was the SourceIP in 9, 235, or 34% of all flow records during the hour. It was found that the Suspicious Host communicated with 5,987 separate DestinationIP's during the hour. These DestinationIP's were distributed around the globe. Further analysis of the ports targeted by more than 10,000 flows revealed that almost all traffic from SourceIP's that targeted the Suspicious Host as the DestinationIP was using protocol 17 and destination port 27015 (Table 1). Also, a significant amount of the traffic to and from Suspicious Host was directed at university campuses in the United States and consumer ISP's around the world. Although not elaborated on herein due to space limitations, the reader should note the somewhat uniform distribution of flows using ports 27,000 - 27,005 and 27,010 in Table 1.

Figure 2

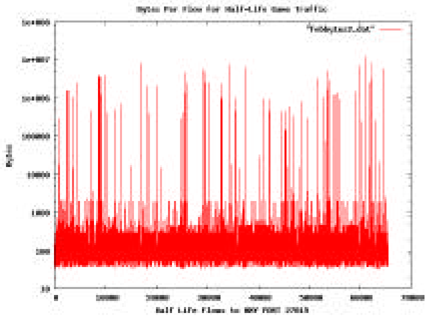


Figure 2 shows the bytes per flow for traffic where either source or destination port was set to 27015.The slice presented in Figure 2 covers a period of approximately 2 hours, or approximately 65,000 consecutive flow records. The repeating pattern of low, medium and high byte volumes is indicative of the presumed application protocol. This behavior is manifest as low followed by medium volume for all external SourceIP's followed by regular high volume by a smaller set of SourceIP's. Finally, the Suspicious Host was identified as an experimental development machine that had been part of a development and testing project in the previous year. Although it was still connected to the network it was not supposed to have any active users. The known facts about the Suspicious Host are summarized below.

Suspicious Host
Responsible for 56% of Network Non-Port 80 Byte Volume
Responsible for 34% of Network Non-Port 80 Flow Volume
Some Preference for University Campuses and Consumer ISP's
Primarily uses UDP Port 27015
Should Have Little or No Traffic

A search of the Internet revealed that (with the exception of the last feature) this is the behavior pattern of a server involved the Half-Life$^{tm}$ on-line first person shooter game. It appeared that at some time in the past this host had been compromised and was now being used as part of a worldwide on-line gaming community.

It is important to point out that since the experimenter had no access to the actual machine or payload data this conclusion is simply conjecture. However, it is one with which the author is exceedingly comfortable. Furthermore, it is important to point out that Valve Software, the maker of Half-Life$^{tm}$ is a legitimate company that would never knowingly allow its products to be part of an unauthorized network compromise. Indeed, in these circumstances, companies such as Valve Software are as much a victim as the owner of the compromised network. The Half-life$^{tm}$ game, like other such on-line games, contains Client and Server software. The player installs the client software on their own machine and then searches for an available server. These servers are run by other players or by server hosts. This particular game uses a third component known as Meta-Servers [4] or Master Servers [5] which provide a list of known game servers. Without access to the suspect machine to search for installed software we are left to speculate as to whether a Server or Meta-Server was installed. Further analysis of the traffic pattern of the compromised machine and comparison to the traffic that one would see from actual known Servers and Meta-Servers would most likely resolve this ambiguity. Such a comparison does not form part of this analysis.

On February 13, the owner of the compromised host was advised of the infection. The owner indicated his intention to immediately bock subsequent access to the machine by the gaming community. The author recommended against this action. However, the machine owner decided to proceed with shutting down access. This action proved unsuccessful and gaming traffic continued for another month.

## 2. The Search for a Behavioral Signature

An investigation was undertaken to attempt to discover a TCP signature that could be associated with game traffic.

### 2.1. Separating Normal from Infected Traffic

Each action listed below is accompanied by an example Silktools Command used to achieve the desired result. In order to discover an un-infected model of the Network traffic it was necessary to remove the traffic which could be attributed to those hosts involved in the gaming traffic, thus creating an artificial normal traffic sample. This normal traffic would then be compared to the infected traffic in an attempt to find distinguishing characteristics. To achieve this separation, a filtering of the data was done to extract all SourceIP's that specified either the source or destination port as 27005, 27014 or 27015 within a 24-hour period when infection was known to be present. The resultant file was labeled as the half-life traffic (**rwfilter --aport=27005,27014,27015 --pass=hltraffic.f out* % out* will open each flow record file for the day in sequence).**

A set of unique SourceIP's was then created using the half-life traffic file as input and labeled hlsipfile.set (**rwset --sip-file=hlsipfile.set hltraffic.f).**

We now had a list of unique SourceIP's allegedly involved in the game traffic. To remove the ambiguity of action which is naturally associated with UDP traffic it was decided to see how much (if any) TCP traffic these game SourceIP's participated in, and what was the nature of that traffic. A set of unique SourceIP's for all TCP traffic was created. This was done in two steps. First the TCP traffic was isolated from the rest (**rwfilter - -proto=6 - -pass=tcptraffic.f out* ).** Next a set of unique SourceIP's was created (**rwset - - sip-file=tcpsipfile.set tcptraffic.f).**

The Game SourceIP's involved in TCP traffic are given by the intersection of the two previous sets and placed in the file hltcp.set (**setintersect - -add-set=hlsipfile.set - -add-set=tcptraffic.f - -set-file=hltcp.set).**

Upon completion hltcp.set contained only four unique SourceIP's, one of which was the Suspicious Host. The other three were not within the address space of the Network. By removing traffic involving these SourceIP's from the total TCP traffic we are left with a file of TCP transactions where the participating game players and the Suspicious host have been removed. This TCP traffic we label as normal, or without-infection. Figure 3 shows two images of the data. The figure on the left shows a small area of the plot of destination ports versus Bytes per flow in the presence of the infection. The image on the right of the figure shows the same area of the solution space when the infecting traffic has been removed.

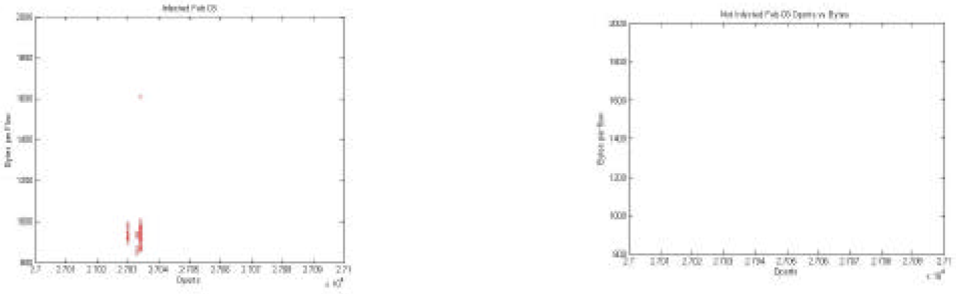


Figure 3

From the two images one can discover a set of TCP traffic contained in the Destination Port range 27,030, 27,033 and 27,034 with Bytes Per Flow sizes ranging from the low 800's to slightly more than 1000 with a noticeable outlier at approximately 1600. This traffic is absent in the artificial normal data set. This traffic was labeled as possible game Signature Traffic. It was originally the author's hypothesis that this data set represents the possible presence of a Game Server in a network (i.e. a signature). This hypothesis has yet to be examined fully and tested.

### 3. Future Work

Work on this dataset will continue in August 2006, at which time further searching for a TCP signature indicative of the administrative layer of the game network will be sought. This activity will expand the search to include Port 80 and Null traffic. In addition, the author hypothesizes that the loss of the game server from the network will create a continuing repeating pattern of attempted logins by players. The author has labeled this type of traffic as SCAR traffic, for Severed Connection Anomalous Records. This Scar traffic may indicate the recent presence of a game server (unauthorized) on a network.

## References

[1] SilkTools, http://silktools.sourceforge.net/, as accessed March 28, 2006.

[2] Valve Software, http://www.half-life.com as accessed on March 28, 2006

[3] CISCO IOS Netwflow, http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html, as accessed on March 28, 2006.

[4] Weaver, N, Paxson, V., Staniford, S., Cunningham, R. "Large Scale Malcious Code: A Research Agenda", The International Computer Science Institute (ICSI), a DARPA Sponsored Report, 2003.

[5] HL Master Game Server configuration,http://hlmaster.sourceforge.net/documentation/hlmaster.gameserver.php as accessed on March 31, 2006.

# A Traffic Analysis of a Small Private Network Compromised by an On-line Gaming Host

Ron McLeod, BCSc, MCSc.
Director - Corporate Development Telecom Applications Research Alliance
Doctoral Student, Faculty of Computer Science, Dalhousie University

# Abstract
## (Abridged)

In the early months of 2006 a small private network (the Network) suffered a Noticeable degrading of its network performance. A network traffic capture and analysis was conducted and used to investigate the network performance issues. This paper presents partial results of that analysis. During the first analysis of the captured data it was discovered that the Network contained a host that had been compromised at some time in the past and was currently being used to support the on-line gaming activity of over 174,000 distinct player source addresses around the globe. The initial finding was the result of a manual investigation of unusual time and volume traffic spikes from arbitrarily chosen time slices. Subsequent work was conducted on searching for a traffic signature which could be representative of the presence of the Game such that future discovery of Game activity could be automated. Gaming traffic is predominantly UDP traffic of high byte volumes, typically targeted at a given range of destination ports. This analysis also searches for a specific TCP traffic pattern that is suggestive of a Game signature. Network traffic patterns that emerge after access to the compromised host has been closed are labeled as SCAR traffic, for **S**evered **C**onnection **A**nomalous **R**ecords

# Presentation Outline

- Summary of the event
- A UDP Profile of the Infection
- The Search for a TCP Signature
- The Search for Residual Traffic (SCAR)
- Concluding Remarks

# Event Chronology

- A Traffic Capture was initiated on February 3.
- On February 11 the first slice of data was extracted for analysis.
- On February 13 a Game Server was discovered on a compromised host.
- For the next 30 days this server supported the on-line Game playing of over 174,000 unique Source Addresses.
- During this time the traffic to and from the game server accounted for greater than 50% of the traffic byte volume and 34% of the network flows.

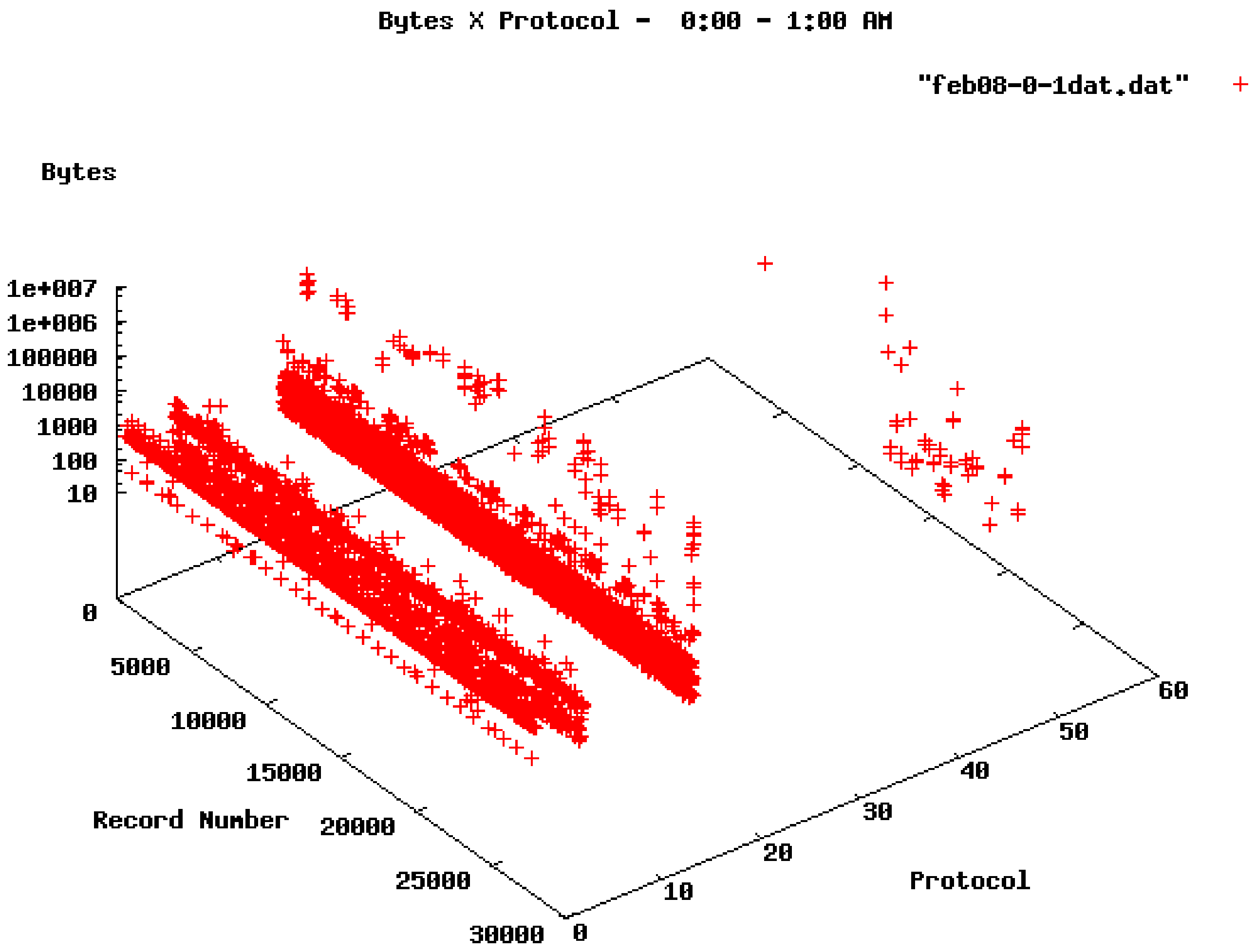
# Network Description

- A Multi-tenant Network consisting of:
  - ~ 40 user assigned hosts, actual number subject to minor fluctuations over time.
  - ~40 special hosts not assigned to individual users. These hosts form parts of various temporary development and experimental environments.
  - Users were apprised that Network flow data was now being captured for experimental and management reasons.
  - Payload data was neither collected nor examined.
  - Analysts did not have access to the content of specific hosts for further investigation.
  - For confidentiality reasons the identity of the Network is not specified in this Presentation.

# First Capture

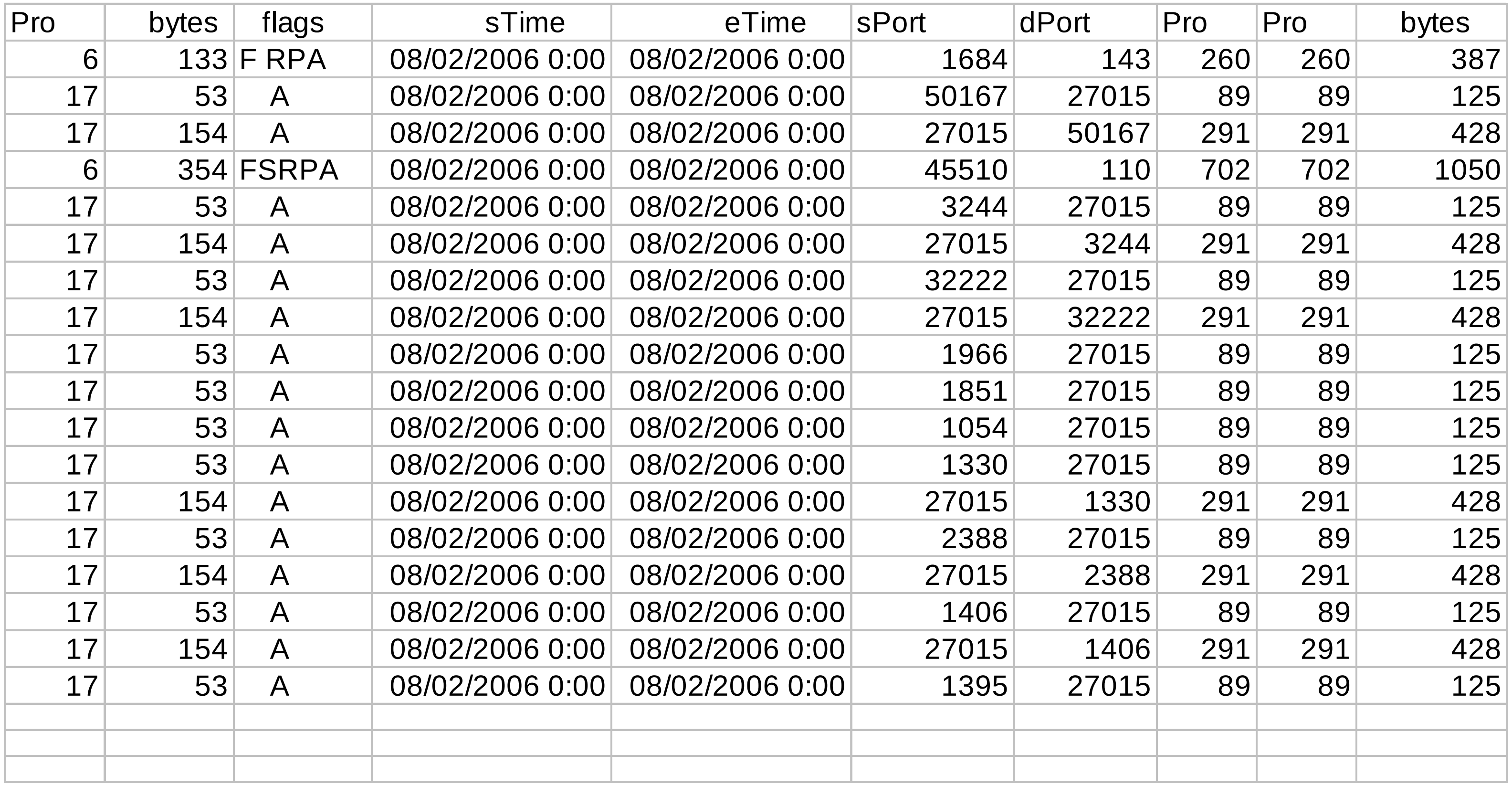
- On February 11 the first sample of network traffic (Slice) was extracted for analysis.
- The time period from midnight to 7:00 AM local time on February 8 was chosen for the first slice.
- This was partially a random choice and partially due to the fact that the author expected minimal traffic volumes during this time. The institution which houses the Network is closed during these hours.
- Only Non-port 80 and Non-Null traffic was initially examined.

# The First Capture Image



# First Traffic Capture Observations

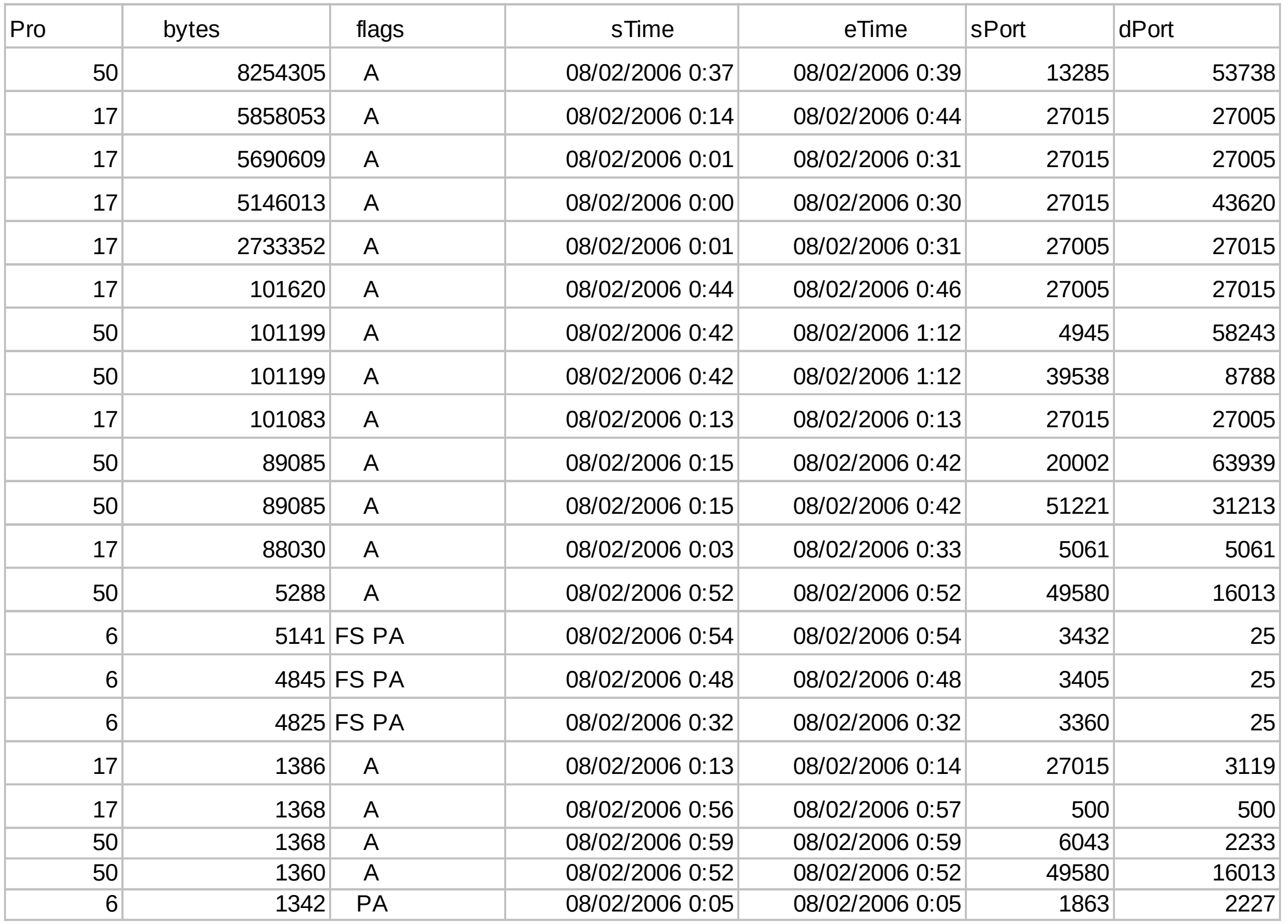
- Protocols are as one would expect (1,6,17,50*,53*).
- Size raised suspicion: 27,000+ records per hour *seemed* large for a network with no active users.



| Pro | bytes | flags | sTime | eTime | sPort | dPort | Pro | Pro | bytes |
|---|---|---|---|---|---|---|---|---|---|
| 6 | 133 | F RPA | 08/02/2006 0:00 | 08/02/2006 0:00 | 1684 | 143 | 260 | 260 | 387 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 50167 | 27015 | 89 | 89 | 125 |
| 17 | 154 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 27015 | 50167 | 291 | 291 | 428 |
| 6 | 354 | FSRPA | 08/02/2006 0:00 | 08/02/2006 0:00 | 45510 | 110 | 702 | 702 | 1050 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 3244 | 27015 | 89 | 89 | 125 |
| 17 | 154 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 27015 | 3244 | 291 | 291 | 428 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 32222 | 27015 | 89 | 89 | 125 |
| 17 | 154 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 27015 | 32222 | 291 | 291 | 428 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 1966 | 27015 | 89 | 89 | 125 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 1851 | 27015 | 89 | 89 | 125 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 1054 | 27015 | 89 | 89 | 125 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 1330 | 27015 | 89 | 89 | 125 |
| 17 | 154 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 27015 | 1330 | 291 | 291 | 428 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 2388 | 27015 | 89 | 89 | 125 |
| 17 | 154 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 27015 | 2388 | 291 | 291 | 428 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 1406 | 27015 | 89 | 89 | 125 |
| 17 | 154 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 27015 | 1406 | 291 | 291 | 428 |
| 17 | 53 | A | 08/02/2006 0:00 | 08/02/2006 0:00 | 1395 | 27015 | 89 | 89 | 125 |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# First Traffic Capture Observations

- Records were then ordered by byte size



| Pro | bytes | flags | sTime | eTime | sPort | dPort |
|---|---|---|---|---|---|---|
| 50 | 8254305 | A | 08/02/2006 0:37 | 08/02/2006 0:39 | 13285 | 53738 |
| 17 | 5858053 | A | 08/02/2006 0:14 | 08/02/2006 0:44 | 27015 | 27005 |
| 17 | 5690609 | A | 08/02/2006 0:01 | 08/02/2006 0:31 | 27015 | 27005 |
| 17 | 5146013 | A | 08/02/2006 0:00 | 08/02/2006 0:30 | 27015 | 43620 |
| 17 | 2733352 | A | 08/02/2006 0:01 | 08/02/2006 0:31 | 27005 | 27015 |
| 17 | 101620 | A | 08/02/2006 0:44 | 08/02/2006 0:46 | 27005 | 27015 |
| 50 | 101199 | A | 08/02/2006 0:42 | 08/02/2006 1:12 | 4945 | 58243 |
| 50 | 101199 | A | 08/02/2006 0:42 | 08/02/2006 1:12 | 39538 | 8788 |
| 17 | 101083 | A | 08/02/2006 0:13 | 08/02/2006 0:13 | 27015 | 27005 |
| 50 | 89085 | A | 08/02/2006 0:15 | 08/02/2006 0:42 | 20002 | 63939 |
| 50 | 89085 | A | 08/02/2006 0:15 | 08/02/2006 0:42 | 51221 | 31213 |
| 17 | 88030 | A | 08/02/2006 0:03 | 08/02/2006 0:33 | 5061 | 5061 |
| 50 | 5288 | A | 08/02/2006 0:52 | 08/02/2006 0:52 | 49580 | 16013 |
| 6 | 5141 | FS PA | 08/02/2006 0:54 | 08/02/2006 0:54 | 3432 | 25 |
| 6 | 4845 | FS PA | 08/02/2006 0:48 | 08/02/2006 0:48 | 3405 | 25 |
| 6 | 4825 | FS PA | 08/02/2006 0:32 | 08/02/2006 0:32 | 3360 | 25 |
| 17 | 1386 | A | 08/02/2006 0:13 | 08/02/2006 0:14 | 27015 | 3119 |
| 17 | 1368 | A | 08/02/2006 0:56 | 08/02/2006 0:57 | 500 | 500 |
| 50 | 1368 | A | 08/02/2006 0:59 | 08/02/2006 0:59 | 6043 | 2233 |
| 50 | 1360 | A | 08/02/2006 0:52 | 08/02/2006 0:52 | 49580 | 16013 |
| 6 | 1342 | PA | 08/02/2006 0:05 | 08/02/2006 0:05 | 1863 | 2227 |

# First Traffic Capture Observations

From this sorting it was discovered that a small group of SourceIPs using protocol 17 appeared to be responsible for a large portion of the traffic bytes.

However, given the size of the database it was not immediately apparent if There was a subset of these hosts that were unusually heavier than the rest.

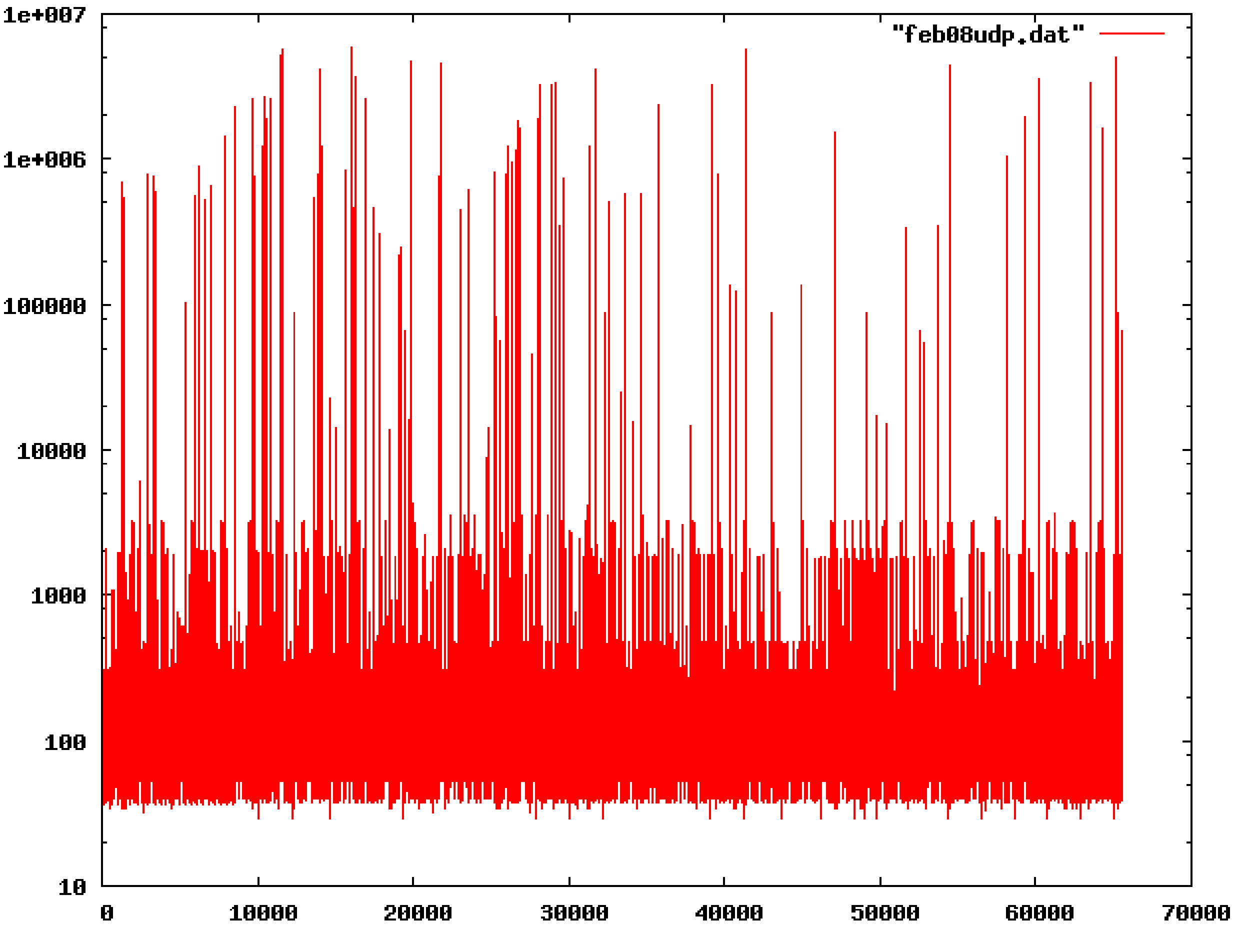
The traffic was then sorted by Source IP and the total bytes over all flow Records were accumulated for each SourceIP.

One SourceIp, labeled Suspicious Host, accounted for more than 56% of the traffic volume in bytes

| | |
|---|---|
| Total Bytes for 12:00 – 1:00AM | 142,129,799 |
| Total Byte Volume for Suspicious Host 12:00 – 1:00AM | 79,865,126 |

# Feb 08 UDP Traffic

Bytes



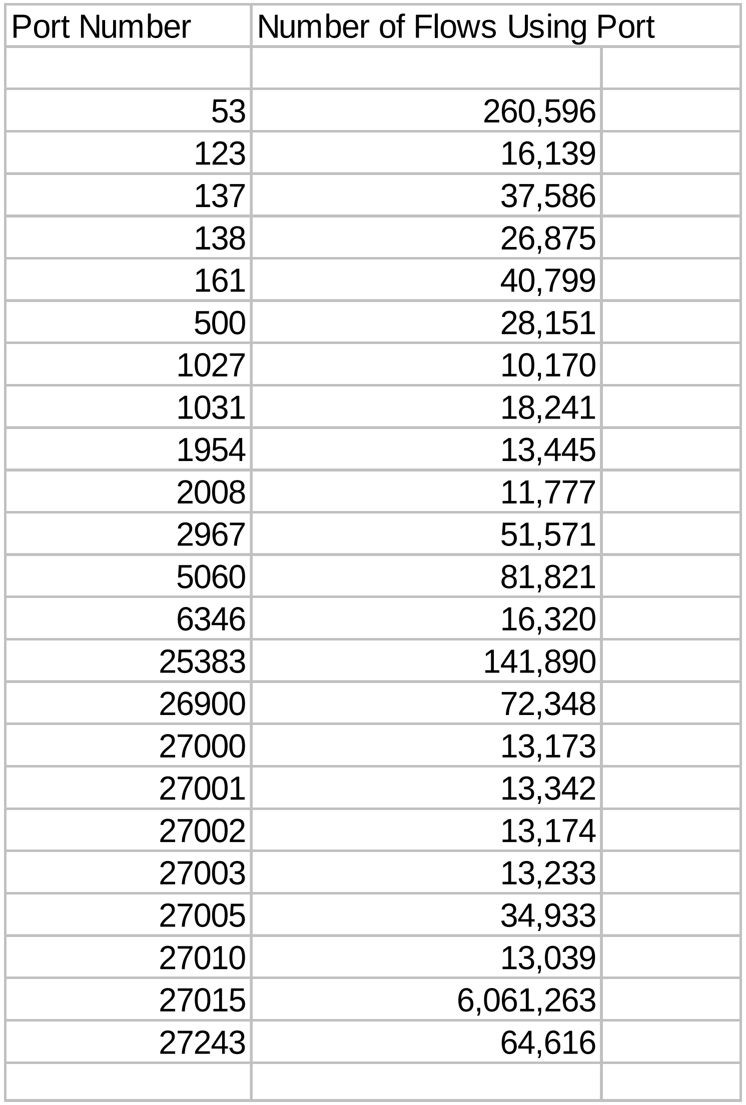
12:00 – 2:00 AM

# First Traffic Capture Observations

The Next Step was to examine the use of UDP Ports.

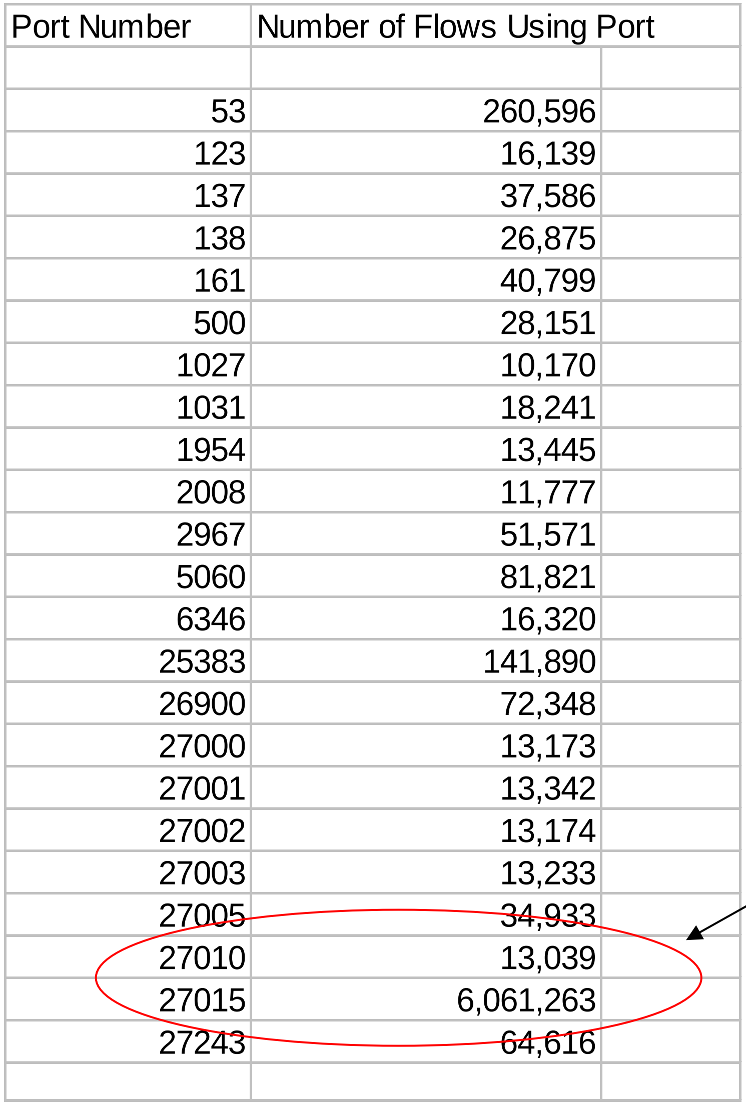
This was done by creating Port Bags and reporting on Key counts greater than 10,000.

# Port Bag For Key Counts > 10,000



| Port Number | Number of Flows Using Port |
|---|---|
| 53 | 260,596 |
| 123 | 16,139 |
| 137 | 37,586 |
| 138 | 26,875 |
| 161 | 40,799 |
| 500 | 28,151 |
| 1027 | 10,170 |
| 1031 | 18,241 |
| 1954 | 13,445 |
| 2008 | 11,777 |
| 2967 | 51,571 |
| 5060 | 81,821 |
| 6346 | 16,320 |
| 25383 | 141,890 |
| 26900 | 72,348 |
| 27000 | 13,173 |
| 27001 | 13,342 |
| 27002 | 13,174 |
| 27003 | 13,233 |
| 27005 | 34,933 |
| 27010 | 13,039 |
| 27015 | 6,061,263 |
| 27243 | 64,616 |

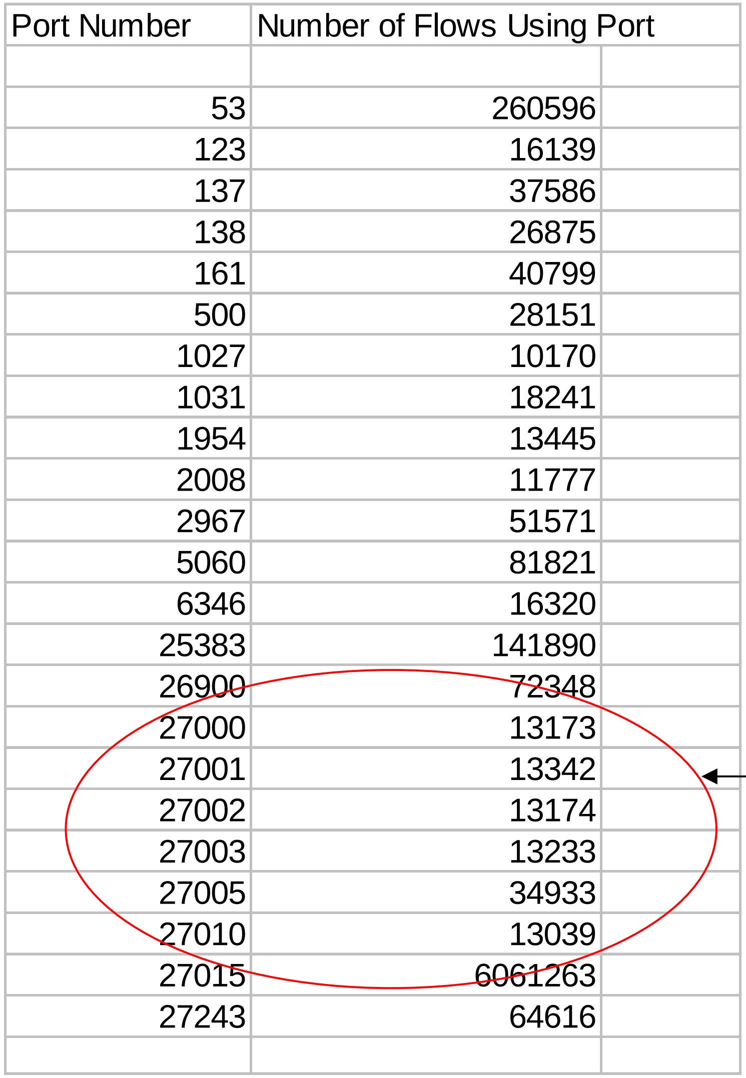
# Port Bag For Key Counts > 10,000



| Port Number | Number of Flows Using Port | |
|---|---|---|
| | | |
| 53 | 260,596 | |
| 123 | 16,139 | |
| 137 | 37,586 | |
| 138 | 26,875 | |
| 161 | 40,799 | |
| 500 | 28,151 | |
| 1027 | 10,170 | |
| 1031 | 18,241 | |
| 1954 | 13,445 | |
| 2008 | 11,777 | |
| 2967 | 51,571 | |
| 5060 | 81,821 | |
| 6346 | 16,320 | |
| 25383 | 141,890 | |
| 26900 | 72,348 | |
| 27000 | 13,173 | |
| 27001 | 13,342 | |
| 27002 | 13,174 | |
| 27003 | 13,233 | |
| 27005 | 34,933 | |
| 27010 | 13,039 | |
| 27015 | 6,061,263 | |
| 27243 | 64,616 | |
| | | |

**Ah hah!**

# Port Bag For Key Counts > 10,000



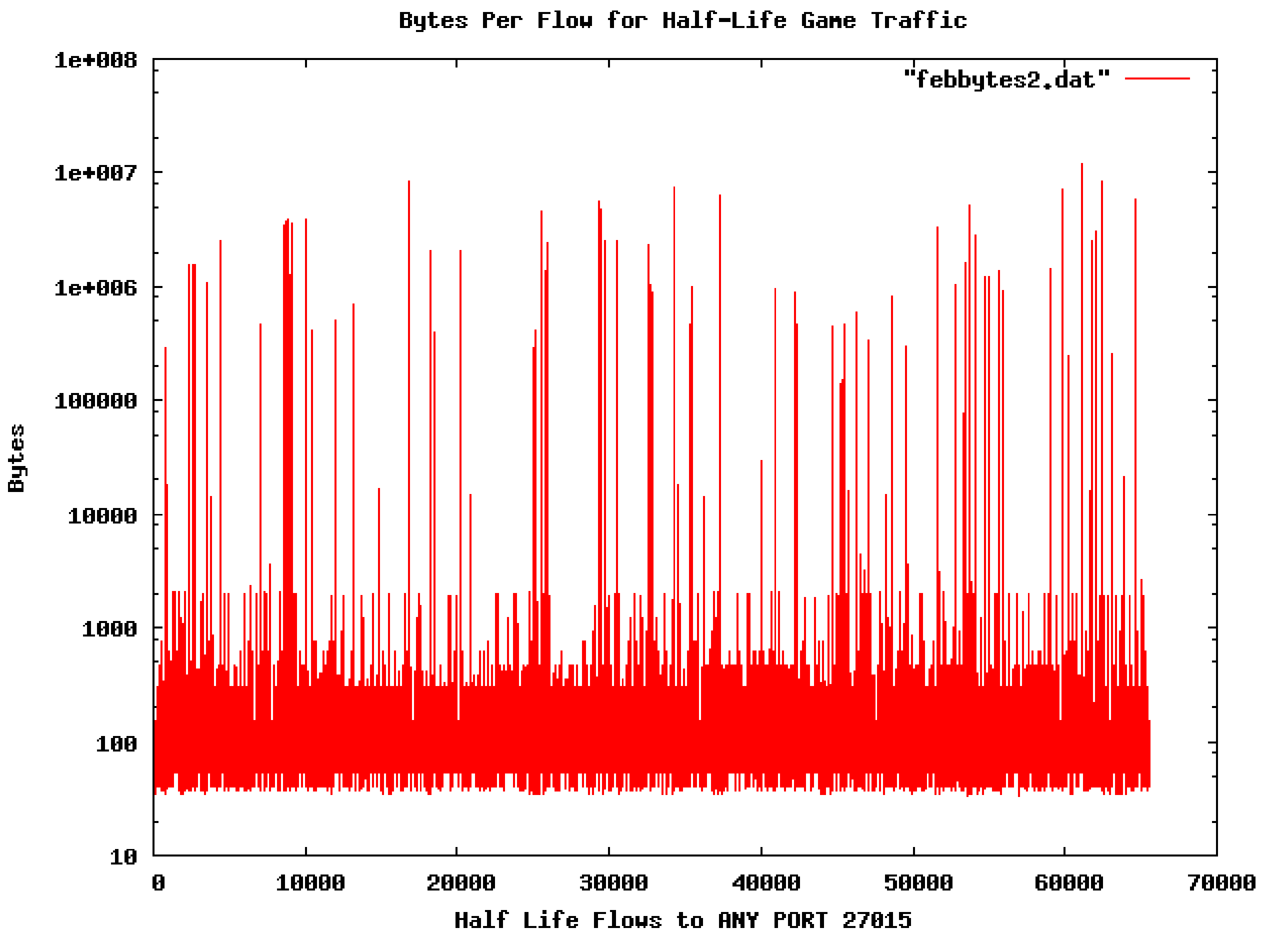
| Port Number | Number of Flows Using Port |
|---|---|
| | |
| 53 | 260596 |
| 123 | 16139 |
| 137 | 37586 |
| 138 | 26875 |
| 161 | 40799 |
| 500 | 28151 |
| 1027 | 10170 |
| 1031 | 18241 |
| 1954 | 13445 |
| 2008 | 11777 |
| 2967 | 51571 |
| 5060 | 81821 |
| 6346 | 16320 |
| 25383 | 141890 |
| 26900 | 72348 |
| 27000 | 13173 |
| 27001 | 13342 |
| 27002 | 13174 |
| 27003 | 13233 |
| 27005 | 34933 |
| 27010 | 13039 |
| 27015 | 6061263 |
| 27243 | 64616 |
| | |

**Also note for future reference**

# First Traffic Capture Observations

- Next we look at the pattern of traffic accessing 27015

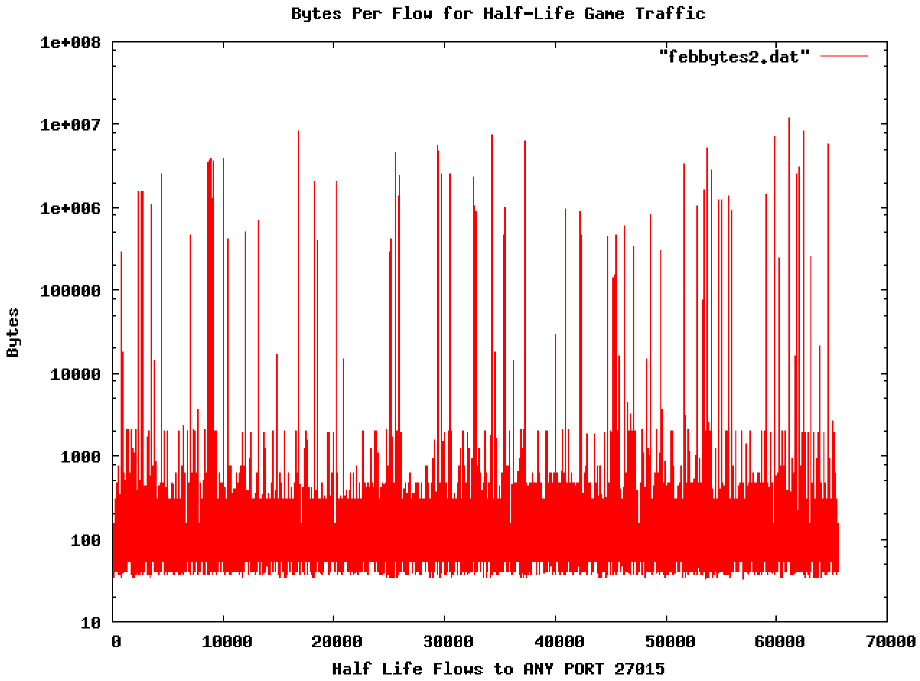
# UDP Traffic Any Port = 27015

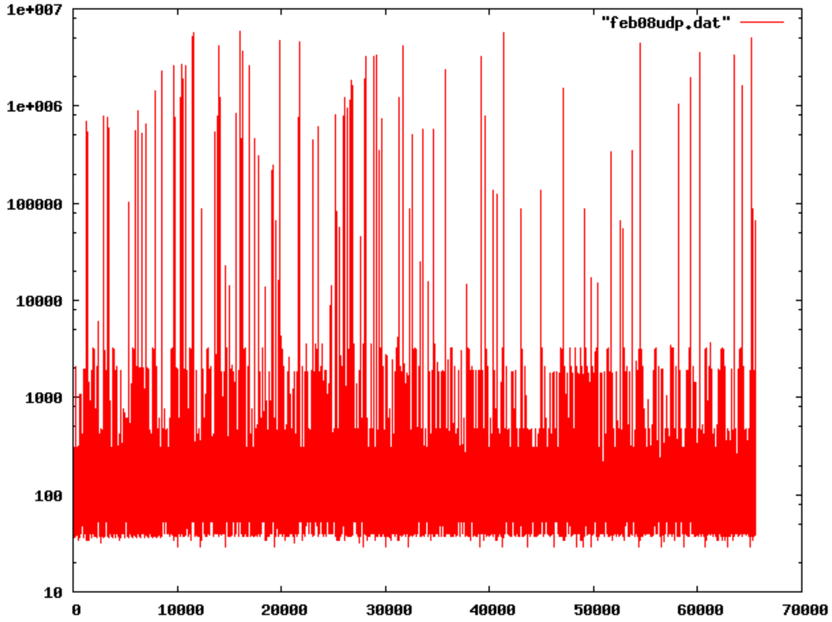


# First Traffic Capture Observations

- Then we do a side by side Comparison to the behavioural pattern with the total UDP traffic.

# Influence of Port 27015 on All UDP





This dominating behavioural pattern was assumed to represent a single application's protocol.

# First Traffic Capture Observations

- Additional information on the characteristics of the Suspicious Host:
  - Suspicious Host was the 34% of all flow records during the hour tested.
  - Suspicious Host communicated with 5,987 separate DestinationIP's during the hour.
  - Almost all traffic from SourceIP's that targeted the Suspicious Host as the DestinationIP was using protocol 17 and destination port 27015
  - A significant amount of the traffic to and from Suspicious Host was directed at a university campuses in the United States and consumer ISP's around the world.
  - Finally, the Suspicious Host was identified as an experimental development machine that had been part of a development and testing project in the previous year. Although it was still connected to the network it was not supposed to have any active users.

# Summary Characteristics of Suspicious Host

•Responsible for 56% of Network Non-Port 80 Byte Volume

•Responsible for 34% of Network Non-Port 80 Flow Volume

•Constant Communication with Thousands of Hosts around the World

•Some Preference for University Campuses and Consumer ISP’s

•Primarily uses UDP Port 27015

•Should Have Little or No Traffic

•WHAT AM I?

# Half-Life$^{tm}$

- An on-line First Person Shooter Game produced by Valve Software
- Based on earlier versions of on-line game engines (Quake) and exists in many variations.

# IMPORTANT DISCLAIMER

It is important to point out that Valve Software, the maker of Half-Life$^{tm}$ is a legitimate company that would never knowingly allow its products to be part of an unauthorized network compromise. Indeed, in these circumstances, companies such as Valve Software are as much a victim as the owner of the compromised network.

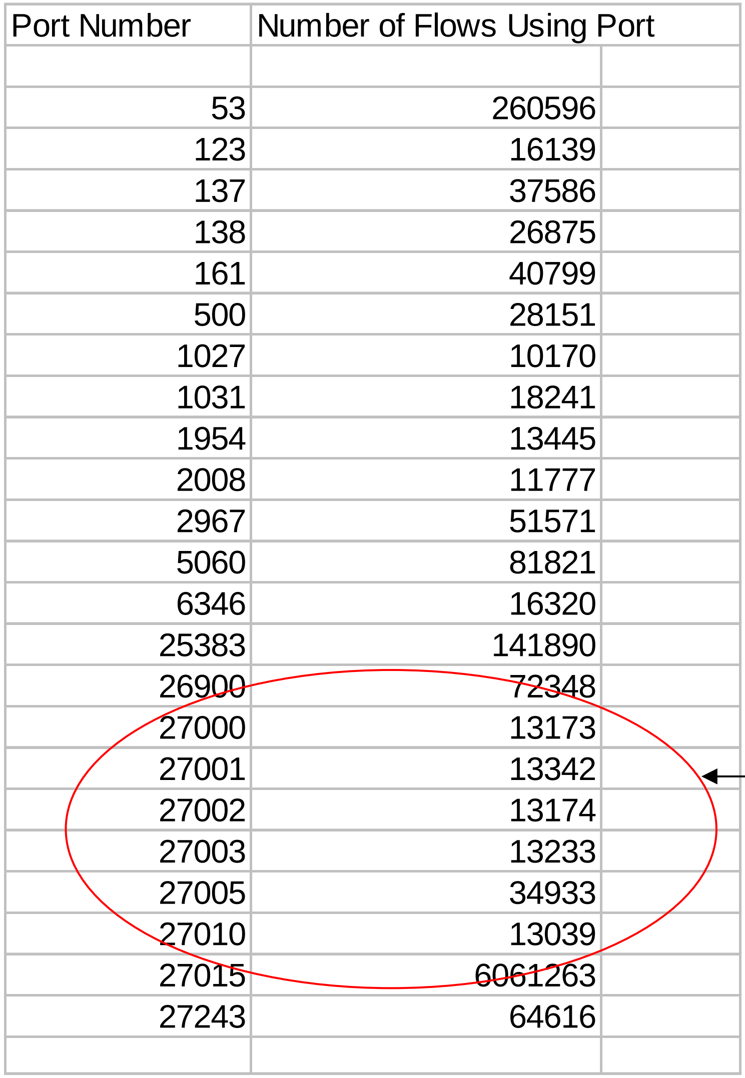
# FURTHER DISCLAIMER

It is important to point out that since the experimenter had no access to the actual machine or payload data this conclusion is simply conjecture.

# Game Characteristics

- Clients communicate with Servers on destination port 27015.
- Game Servers may be initiated by players.
- Meta or Master Servers track available game servers.
- Game servers communicate with Meta servers on UDP port 27010.
- Some TCP Traffic associated with game network management.

# Recall the Presence of Uniform Access in the 27,000 – 27,010 Port Range



| Port Number | Number of Flows Using Port | |
|---|---|---|
| | | |
| 53 | 260596 | |
| 123 | 16139 | |
| 137 | 37586 | |
| 138 | 26875 | |
| 161 | 40799 | |
| 500 | 28151 | |
| 1027 | 10170 | |
| 1031 | 18241 | |
| 1954 | 13445 | |
| 2008 | 11777 | |
| 2967 | 51571 | |
| 5060 | 81821 | |
| 6346 | 16320 | |
| 25383 | 141890 | |
| 26900 | 72348 | |
| 27000 | 13173 | |
| 27001 | 13342 | |
| 27002 | 13174 | |
| 27003 | 13233 | |
| 27005 | 34933 | |
| 27010 | 13039 | |
| 27015 | 6061263 | |
| 27243 | 64616 | |
| | | |

**Also note for future reference**

# Signature is By No Means Unique

- UDP port can be chosen by any application.
- Large byte volume is a relative term
- User demographic (Consumer ISP’s, Campus networks) is determined by looking.
- Would like to find a TCP management signature

# Strategy To Isolate TCP signature

- We know that one exist’s from on-line developer discussions.
- Build a set of Game SIP’s.
- Slice out all TCP traffic.
- Isolate the TCP traffic associated with the Player SIP’s.

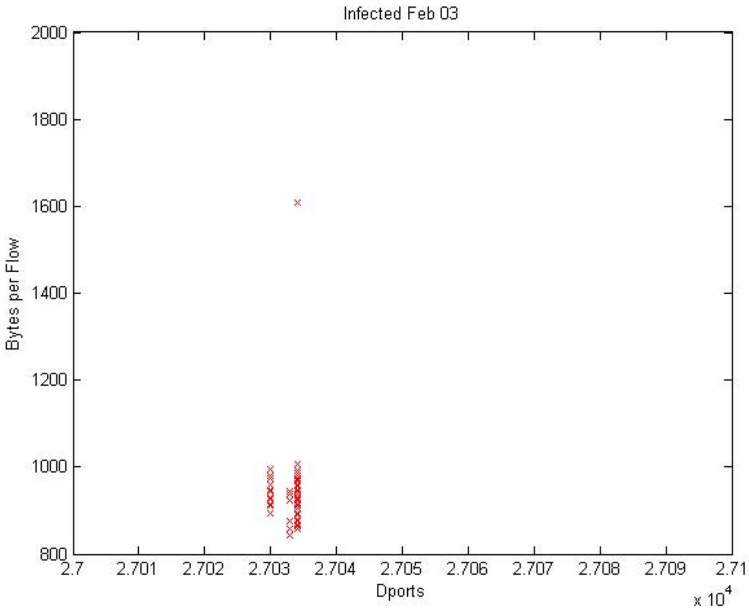
# Strategy To Isolate TCP signature

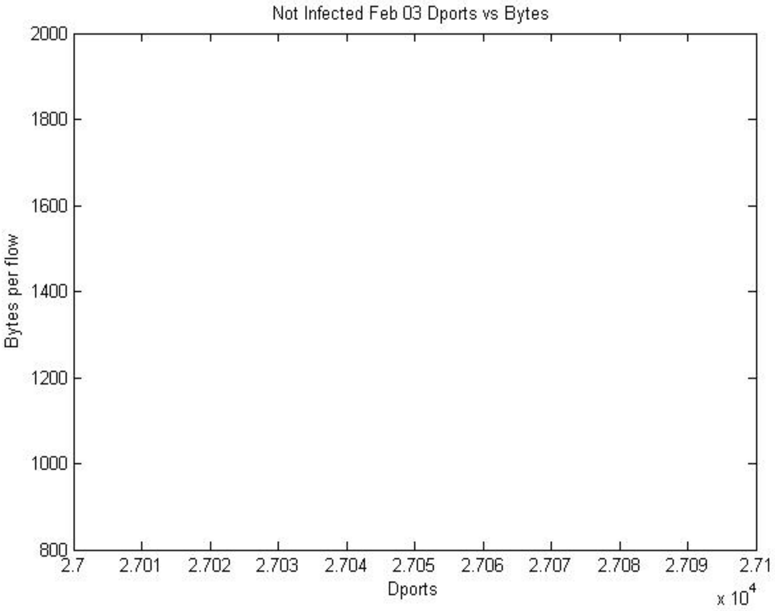
- Build a set of Game SIP’s.
  - Create a game host file:
    - rwfilter  - -aport=27,005,27,014,27015  - -pass=hltraffic.f  out*
  - Create a set of unique IP’s for Game Hosts
    - rwset  - -sip-file=hlsipfile.set hltraffic.f
- Slice out all TCP traffic.
    - rwfilter - -proto=6 - -pass=tcptraffic.f out*
  - Create a set of Unique IP’s for the TCP traffic
    - rwset - - sip-file=tcpsipfile.set tcptraffic.f
- Intersect the sets to get the Game hosts using TCP
    - setintersect  - -add-set=hlsipfile.set - -add-set=tcptraffic.f - -set-file=hltcp.set

# TCP Game Traffic

- Upon completion hltcp.set contained only four unique SourceIP’s, one of which was the Suspicious Host. The other three were not within the address space of the Network.
- Removing these SIP’s from the complete file of TCP traffic created an artificial normal TCP traffic slice
- Comparing the *Artificial Normal Data* to the actual data revealed a distinguishing pattern.

# A TCP Signature?

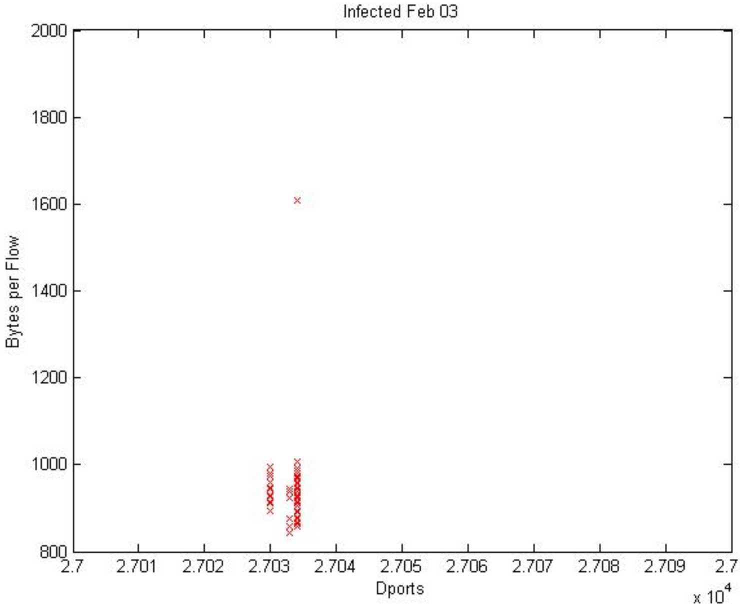


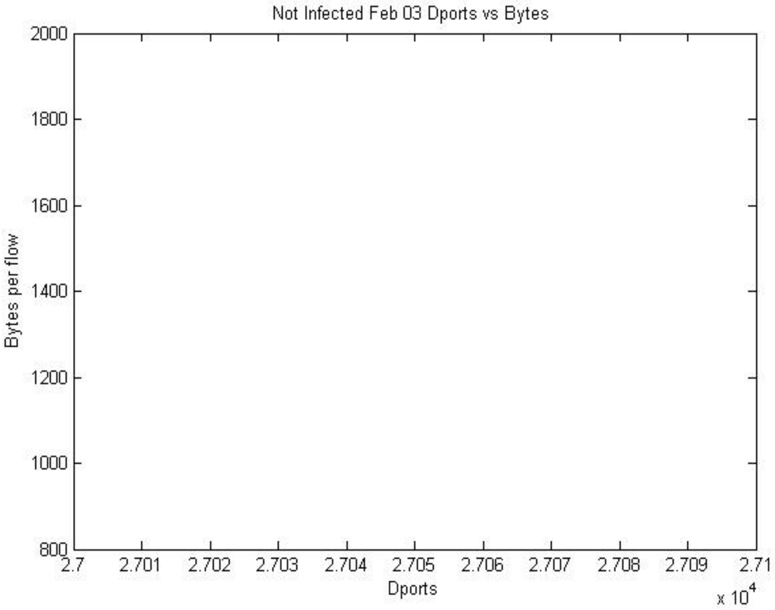


TCP traffic contained in the Destination Port range 27,030, 27,033 and 27,034 with Bytes Per Flow sizes ranging from the low 800’s to slightly more than 1000 with a noticeable outlier at approximately 1600. This traffic is absent in the artificial normal data set.

# A TCP Signature?

## Probably Not





# Be Careful of Assumptions

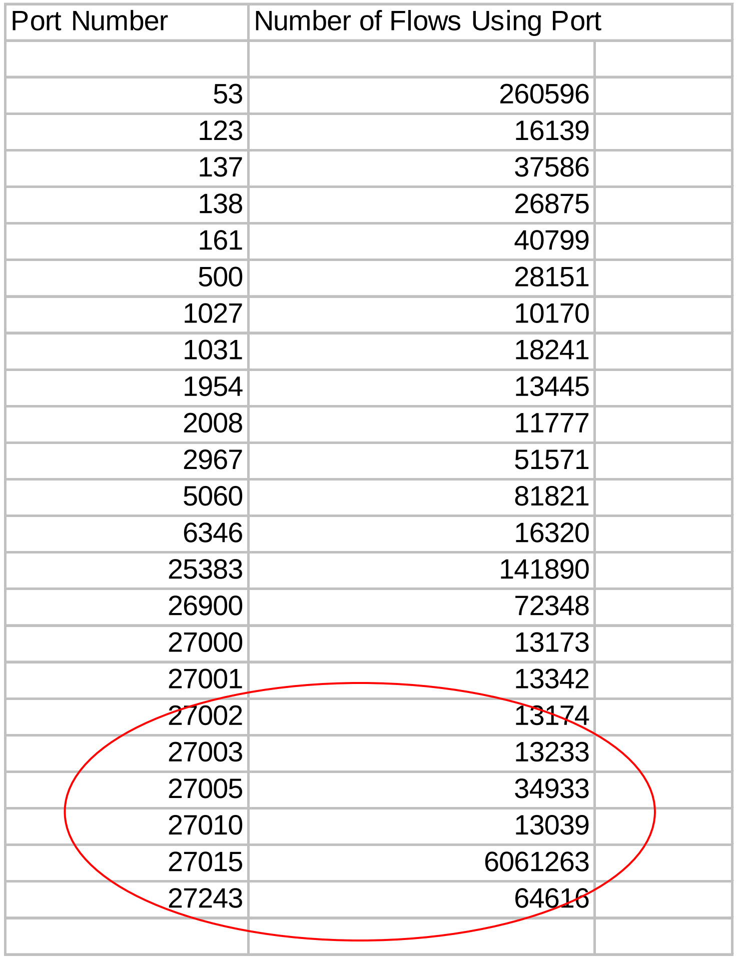
- This Host was not supposed to have any active users.
- At least half of the SourceIP’s creating the TCP Signature were immediately known to the Owner.

# Be Careful of Assumptions

- This Host was not supposed to have any active users.
- At least half of the SourceIP's creating the TCP Signature were immediately known to the Owner.
- However – An on-line discussion mentions

  Server to server communication to a European address range that exists in the data and communication on port 27010

# Port Bag For Key Counts > 10,000



| Port Number | Number of Flows Using Port |
|---|---|
| | |
| 53 | 260596 |
| 123 | 16139 |
| 137 | 37586 |
| 138 | 26875 |
| 161 | 40799 |
| 500 | 28151 |
| 1027 | 10170 |
| 1031 | 18241 |
| 1954 | 13445 |
| 2008 | 11777 |
| 2967 | 51571 |
| 5060 | 81821 |
| 6346 | 16320 |
| 25383 | 141890 |
| 26900 | 72348 |
| 27000 | 13173 |
| 27001 | 13342 |
| 27002 | 13174 |
| 27003 | 13233 |
| 27005 | 34933 |
| 27010 | 13039 |
| 27015 | 6061263 |
| 27243 | 64616 |
| | |

# The Search for a Scar

- Is there a unique traffic signature for a network that previously contained a game server host?
- Is there a residual SCAR in the traffic - Severed Connection Anomalous Records

# The Search for a Scar

- Unfortunately, Game Server was disconnected from the network.
- A search of Null Traffic was conducted which revealed two interesting anomalies

# SCAR Traffic?

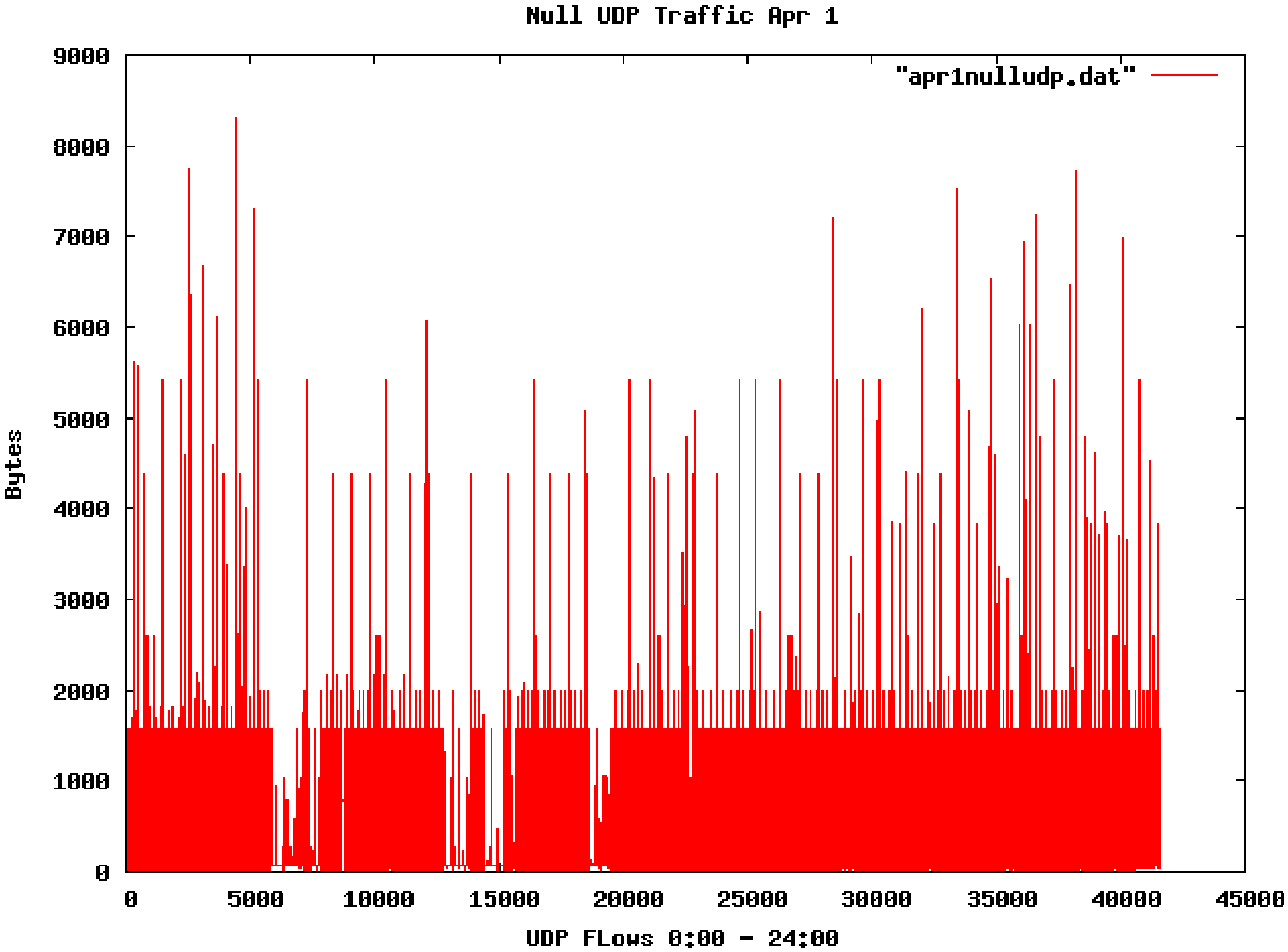
| dPort | pro | bytes | flags |
|---|---|---|---|
| 27015 | 17 | 53 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |

# SCAR Traffic?

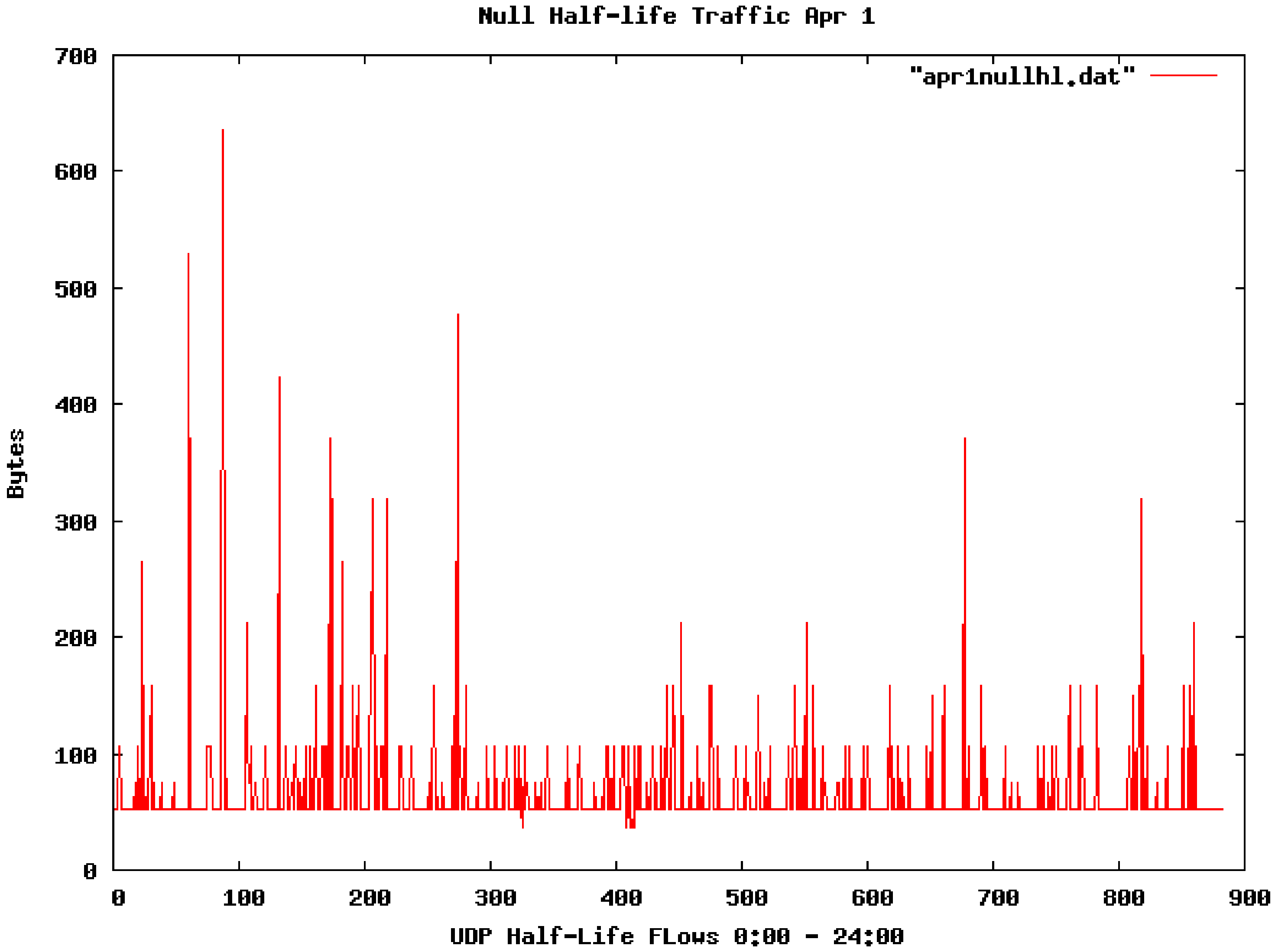
| dPort | pro | bytes | flags |
|---|---|---|---|
| 27015 | 17 | 53 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 106 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |
| 27015 | 17 | 53 | A |

**This same traffic exists in Null while Game server is active???**

# Null UDP Traffic on Apr 1

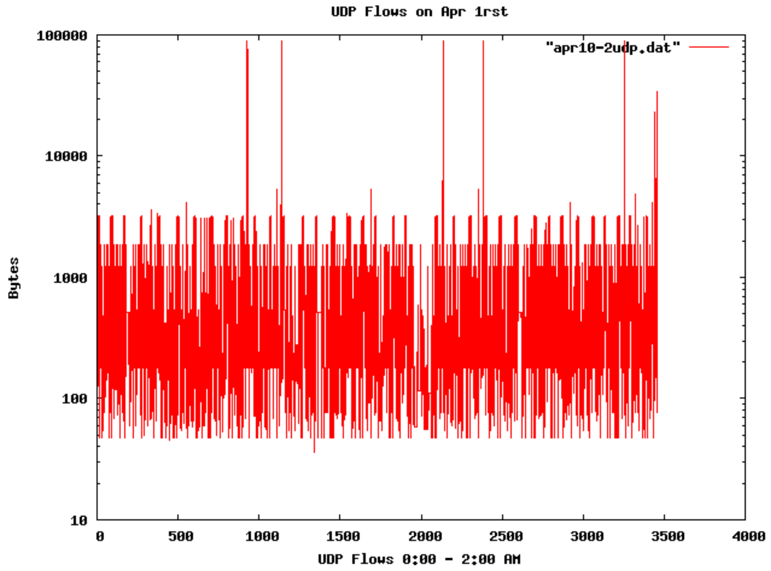


# Null Game Traffic Only on Apr 1

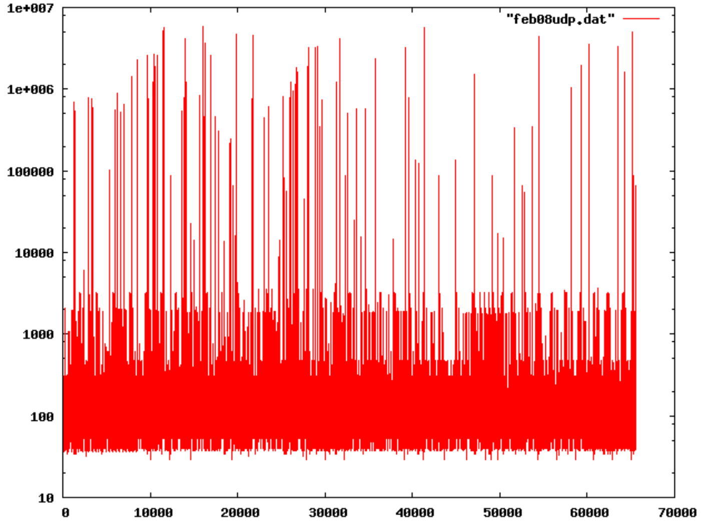


# New Traffic Pattern
# Non-Null Non-Port 80

April 1, 2006



Feb 8, 2006



# Observations on Gaming Traffic

- Much of the existing traffic profiling is aimed at providing a better game experience.
- Consumes considerable Resources.
- Represents a Level 7 WAN Network for Communication.
- Provides a channel to hide Malicious Traffic.

# Future Work

- Anonomize the data so that it might be shared.
- Study the form and distribution of players, servers and meta-servers.
- A search for Management and other signatures continues.
- It was found that a virulent worm entered the network through this server. More on this in session 2.

# Special Thanks

- To Dr. John McHugh for introducing me to a whole new world.
- To TARA for providing me with the support to conduct my research.
- To FloCon for inviting me to talk about both.